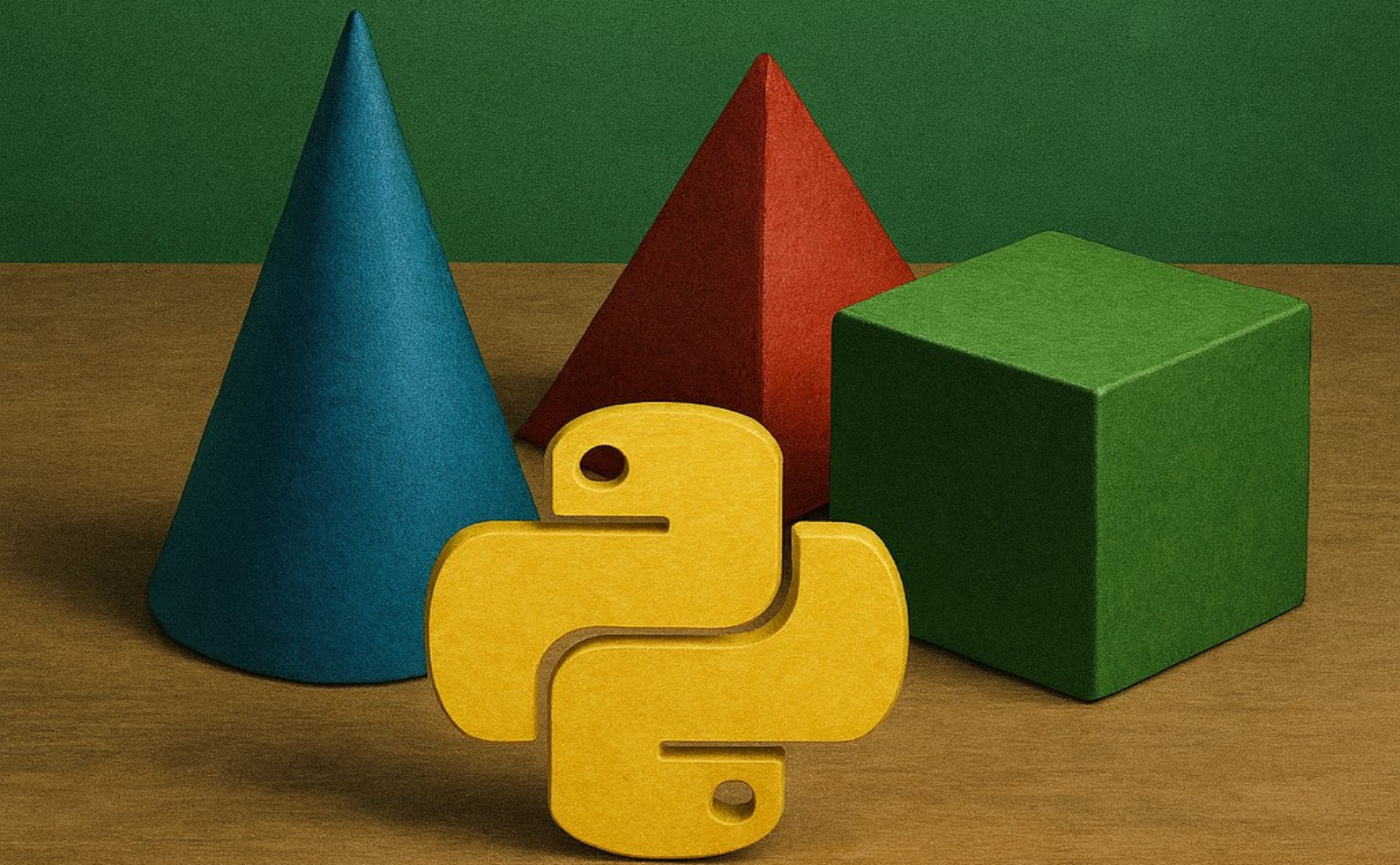
PYTHON

EJERCICIOS V2



LUIS MINGULLÓN PASCUAL

PYTHON
ejercicios v. 2

Luis Minguillón Pascual

COPYRIGHT

La licencia de este libro electrónico es para uso personal . Por lo tanto no puedes revenderlo a otras personas.

Gracias por respetar los derechos de autor.

©Luis Minguillón Pascual
26/10/2025

Prefacio

En la actualidad, la programación se ha convertido en una de las habilidades más valiosas y demandadas en prácticamente todos los ámbitos del conocimiento y la industria. Saber programar no solo implica conocer la sintaxis de un lenguaje, sino también desarrollar la capacidad de pensar de manera lógica, estructurada y analítica para resolver problemas de manera eficiente. Aprender a programar nos permite transformar ideas en soluciones concretas, automatizar procesos, analizar datos y, en última instancia, abrir puertas a infinitas oportunidades profesionales y personales.

Dentro de la amplia variedad de lenguajes de programación disponibles hoy en día, Python ocupa un lugar especial por su simplicidad, claridad y versatilidad. Su sintaxis intuitiva permite que tanto principiantes como expertos puedan centrarse en la resolución de problemas en lugar de perderse en la complejidad del lenguaje. Python se utiliza en campos tan diversos como desarrollo web, inteligencia artificial, análisis de datos, automatización de tareas, videojuegos y muchas otras áreas de innovación tecnológica. Esto convierte a Python en una herramienta ideal para quienes desean iniciar o consolidar su aprendizaje en programación.

El propósito de este libro es ofrecer una colección extensa y práctica de ejercicios que permitan al lector adquirir y consolidar habilidades en Python de manera progresiva. Cada ejercicio ha sido diseñado para estimular el pensamiento lógico, la creatividad y la capacidad de análisis, proporcionando explicaciones claras y soluciones que faciliten la comprensión y el aprendizaje autónomo. La práctica constante es la clave para dominar cualquier disciplina, y la programación no es la excepción. Este libro busca ser un compañero confiable en ese camino de aprendizaje, guiando al lector desde conceptos básicos hasta ejercicios de mayor complejidad, fomentando un aprendizaje sólido y duradero.

Quiero aprovechar estas líneas para expresar mi más profundo agradecimiento a mi familia, cuyo apoyo incondicional ha sido fundamental en la creación de esta serie de libros. Su comprensión, paciencia y ánimo constante han hecho posible que mis ideas y conocimientos se transformen en herramientas que puedan ayudar a otros a crecer y desarrollarse en el mundo de la programación. Este libro no solo es un esfuerzo académico, sino también un proyecto lleno de gratitud hacia quienes han estado a mi lado en cada paso de este camino.

Espero que este libro inspire a sus lectores a explorar, experimentar y aprender, y que sirva como una guía útil y motivadora para todos aquellos que deseen adentrarse en el apasionante mundo de la programación en Python.

Luis Minguillón Pascual

ÍNDICE

I. Ejercicios Python: Variables y operadores (1-100)

A. Operaciones Fundamentales y Cálculos

- **Suma de dígitos** de un número entero (positivo).
- **Contar cuántos dígitos** tiene un número entero.
- **Invertir un número** entero.
- Calcular el **máximo y mínimo** entre tres números.
- Calcular la **media (promedio)** de N números introducidos por el usuario.
- Calcular la **potencia a^b** con exponente no negativo (sin usar **).
- **Redondear un número a su parte entera** (truncamiento hacia cero) sin usar `int()`.
- Calcular la **diferencia absoluta** entre dos números.
- Calcular el **cuadrado y el cubo** de un número.
- Calcular el **perímetro de un rectángulo**.
- Calcular el **promedio de tres números**.
- Calcular el **resto de una división** sin usar el operador %.
- Calcular la **potencia de 2 más cercana** a un número dado.
- Calcular el **producto** de los números del 1 al N.
- Calcular la **suma de cuadrados** de los números del 1 al N.
- Calcular la **suma de cubos** de los números del 1 al N.
- Calcular el **factorial** de un número usando `while`.

B. Divisibilidad y Propiedades Numéricas

- Comprobar si un número es **primo** (método sencillo).
- Mostrar la **tabla de multiplicar** de un número (del 1 al 10).
- Mostrar los **divisores** de un número.
- Comprobar si dos números son **múltiplos** entre sí.
- Calcular el **MCD** (máximo común divisor) usando el algoritmo de Euclides.
- Calcular el **MCM** (mínimo común múltiplo) usando MCD.
- Determinar si un número es **capicúa** (palíndromo numérico).
- Comprobar si un número es **divisible entre 5**.
- Comprobar si un número es **divisible entre 7**.
- Verificar si un número **termina en 7**.
- Verificar si un número es **múltiplo de otro**.
- Comprobar si un número es **múltiplo de 4 o de 6**.
- Verificar si un número es **divisible entre 2, 3 o 5**.
- Verificar si un número es **divisible entre 2 y 3** al mismo tiempo.
- Calcular el **número de cifras pares** en un número entero.
- Determinar si un número tiene **tres dígitos**.

C. Secuencias y Acumulaciones

- Generar y mostrar los **N primeros números pares**.

- Generar y mostrar los **N primeros números impares**.
- Calcular la **suma de los números del 1 al N**.
- Calcular la **suma de los números pares** entre 1 y N (inclusive).
- Calcular la **suma de los números impares** entre 1 y N.
- Mostrar los **múltiplos de 3** entre 1 y 50.
- Calcular el **cuadrado de los números del 1 al 10**.
- Mostrar los **números del 1 al 20**.
- Mostrar los **números del 20 al 1** en orden descendente.
- Calcular la **suma de los pares** entre 1 y 100.
- Calcular la **suma de los impares** entre 1 y 100.
- Mostrar los **múltiplos de 4** entre 1 y 50.
- Contar cuántos números del 1 al 100 son **divisibles entre 5**.
- Calcular el **promedio de los números del 1 al 10**.
- Mostrar los **números comprendidos entre dos valores** dados por el usuario.
- Calcular el **promedio de N números** hasta ingresar 0.
- Mostrar los **10 primeros múltiplos** de un número.
- Mostrar los **números pares** entre dos valores dados.
- Calcular la **suma de los números** comprendidos entre dos enteros.
- Mostrar los **números impares** entre 1 y 30.
- Calcular el **cubo de los números del 1 al 10**.
- Calcular el **promedio de los números pares** entre 1 y N.
- Mostrar los números del 100 al 200 que sean **divisibles entre 7**.
- Mostrar los números comprendidos entre 50 y 100 que sean **múltiplos de 9**.
- Mostrar la **suma de los múltiplos de 5** menores que 50.
- Mostrar el **cuadrado de los números pares** del 1 al 20.
- Mostrar el **cubo de los números impares** del 1 al 15.
- Calcular la **suma de todos los números múltiplos de 3 y 5** entre 1 y 100.
- Calcular la **suma de los números pares y la suma de los impares** entre 1 y N.
- Mostrar los números que son **divisibles entre 3 pero no entre 5** entre 1 y 50.
- Mostrar los números entre 1 y 100 que sean **divisibles por 4 y por 6**.
- Calcular la **suma de los múltiplos de 10** entre 10 y 100.
- Calcular el **producto de los números pares** del 1 al 10.
- Mostrar los **números del 1 al N en orden descendente**.
- Calcular la **suma de los números de 1 a N que sean múltiplos de 4**.
- Mostrar los **números del 1 al N y su cuadrado**.
- Mostrar los **números del 1 al N y su cubo**.
- Mostrar los **múltiplos de 7** entre 1 y 100.
- Mostrar los números del 1 al 20 que sean **pares y múltiplos de 3**.
- Calcular el **promedio de los números impares** del 1 al N.
- Mostrar los **primeros N números positivos**.
- Calcular la **suma de los primeros N números impares**.
- Mostrar los números del 1 al N que sean **múltiplos de 2 o de 5**.
- Calcular la **suma de los primeros N números pares**.

II. Ejercicios Python: Cadenas de Texto (1-100)

A. Conversión y Formato

- Mostrar una cadena introducida.
- Calcular la **longitud** de una cadena.
- Convertir a **mayúsculas** (`.upper()`).
- Convertir a **minúsculas** (`.lower()`).
- **Capitalizar** un texto (`.capitalize()`).
- Convertir la **primera letra de cada palabra en mayúscula** (`.title()`).
- **Repetir** una palabra varias veces (operador `*`).
- Quitar **espacios al principio y al final** (`.strip()`).
- Quitar solo los **espacios del principio** (`.lstrip()`).
- Quitar solo los **espacios del final** (`.rstrip()`).
- **Reemplazar** palabras por otra.
- **Reemplazar espacios por guiones**.
- **Reemplazar cada espacio por una coma**.
- **Sustituir las letras por su posición numérica** (a=1, b=2...) usando `ord()`.
- **Intercalar guiones** entre letras (`.join()`).
- **Insertar un carácter** cada N cantidad de letras.
- **Sustituir las vocales por asteriscos**.
- **Sustituir las vocales por números** (a=1, e=2...).
- **Sustituir todas las letras “a” por “@”**.
- **Convertir la frase a mayúsculas/minúsculas/capitalizado**.

B. Análisis y Conteo

- **Contar una letra específica** (`.count()`).
- **Contar vocales**.
- **Contar palabras** en una frase (`.split()` y `len()`).
- **Contar espacios** en una frase (`.count(" ")`).
- **Contar consonantes**.
- **Contar cuántas veces aparece cada letra** (usando `set`).
- **Contar letras y números** en un texto.
- **Contar cuántas veces aparece una palabra**.
- **Contar cuántas letras hay en total sin espacios**.
- **Contar cuántas letras tiene la frase** (incluyendo espacios).

C. Filtrado y Extracción

- **Separar las letras** de una palabra (imprimir una por línea).
- Mostrar los **tres primeros caracteres**.
- Mostrar los **tres últimos caracteres**.
- **Eliminar todas las vocales**.
- Mostrar el **carácter central** de una palabra.
- **Eliminar todos los signos de puntuación**.
- **Extraer números** de un texto.

- **Eliminar números** de un texto.
- Mostrar **solo las letras mayúsculas** de una frase.
- Mostrar **solo los dígitos pares** de un texto.
- Mostrar **solo los caracteres alfabéticos**.
- Mostrar **solo los caracteres únicos** del texto.
- **Extraer únicamente los números** que aparecen dentro del texto.

D. Orden y Estructura

- **Unir dos cadenas** (concatenación `+`).
- **Convertir una frase en lista de palabras** (`.split()`).
- **Unir lista de palabras** en una sola cadena (`.join()`).
- **Invertir una cadena** (`[::-1]`).
- **Mostrar las palabras ordenadas** alfabéticamente.
- **Invertir el orden de las palabras** de una frase (`.reverse()`).
- **Dividir un texto en frases** separadas por puntos.
- **Convertir una cadena en una lista de caracteres**.
- **Unir lista de caracteres** en una cadena.
- **Mostrar letras alternas** de una palabra (`[::2]`).
- **Mostrar el texto en forma de triángulo** (creciente/invertido).

E. Validación y Búsqueda Lógica

- Verificar si **contiene una palabra** (`in`).
- Verificar si una cadena **empieza con una letra** (`.startswith()`).
- Comprobar si una palabra es **mayúscula/minúscula** (`.isupper()`/`.islower()`).
- Verificar si una cadena contiene **solo letras** (`.isalpha()`).
- Verificar si una cadena contiene **solo números** (`.isdigit()`).
- Comprobar si una palabra es un **palíndromo**.
- Verificar si una **frase es palíndroma** (ignorando espacios/mayúsculas).
- Verificar si una cadena **está vacía**.

III. Ejercicios Python: Estructuras de Datos, Listas (1-100)

A. Creación, Acceso y Modificación

- Crear una lista básica.
- **Agregar** elemento al final (`.append()`).
- Mostrar la **longitud** (`len()`).
- Mostrar el **primer y último elemento** (índices `` y `[-1]`).
- Eliminar un elemento por su valor (`.remove()`).
- **Insertar** un número en una posición específica (`.insert()`).
- Eliminar el último elemento (`.pop()`).
- Eliminar todos los elementos (`.clear()`).
- **Intercambiar el primer y último elemento**.
- Mostrar elementos que están en **posiciones pares**.

- **Reemplazar los números negativos por cero**.
- **Reemplazar los números impares por cero**.
- **Reemplazar los números mayores que 10 por 10**.
- **Reemplazar los números mayores que 10 por su mitad**.

B. Iteración, Búsqueda y Propiedades

- **Ordenar** una lista (`.sort()`).
- **Invertir el orden** (`.reverse()`).
- **Contar** cuántas veces aparece un número (`.count()`).
- Encontrar el **número máximo/mínimo** (`max()`/`min()`).
- Verificar si un elemento **existe** (`in`).
- Mostrar todos los elementos usando un **bucle `for`**.
- Mostrar el **índice** de un elemento (`.index()`).
- Contar cuántos elementos son **mayores que 5**.
- **Eliminar duplicados** (creando una nueva lista sin `set`).
- Mostrar los **elementos mayores que el promedio** de la lista.
- Separar los **pares de los impares** en dos listas distintas.
- Contar cuántas palabras **empiezan con la letra "p"**.
- Mostrar el **segundo elemento más grande**.
- Mostrar la **palabra más larga/corta** de una frase.
- Mostrar la **suma de los elementos en posiciones impares**.

C. Transformación y Conversión

- **Copiar** una lista (`.copy()`).
- **Sumar todos los números** (`sum()`).
- **Multiplicar cada elemento por 2**.
- **Unir dos listas** (`+`).
- **Repetir** los elementos de una lista (`*`).
- Convertir una cadena de texto en una **lista de caracteres**.
- Convertir una lista de números en una **lista de cadenas**.
- Crear una lista con **números del 1 al 10 usando `range()`**.
- Convertir una lista de números en una lista con sus **cuadrados**.
- Mostrar la lista en formato de **cadena separada por comas** (`.join()`).
- Suma elemento a elemento **dos listas de igual tamaño**.
- Crear una lista con los **caracteres sin repeticiones**.
- Mostrar los elementos **multiplicados por su índice** (`enumerate()`).
- Crear una lista con los **primeros 10 números cuadrados**.
- Contar cuántos elementos son **cadenas de texto** (`isinstance()`).
- Lista con números al **cuadrado solo si son pares**.
- Lista con los **primeros 10 números impares**.
- Lista con los elementos de otra lista **multiplicados por 3**.
- Lista con la **longitud de cada cadena**.
- Lista con las palabras que tengan **más de 3 letras**.
- Lista con los **números negativos convertidos en positivos** (`abs()`).

- Lista con los elementos de otra lista **repetidos dos veces**.
- Lista con los elementos en **orden inverso usando slicing** (`[::-1]`).
- Lista con los números **múltiplos de 5**.
- Lista con la **primera letra de cada palabra**.
- Lista con los **números divididos entre 2**.
- Lista con los elementos **positivos primero y los negativos después**.

IV. Ejercicios Python: Conjuntos (1-100)

A. Creación y Manipulación

- Crear un **conjunto** con números.
- Crear un conjunto a partir de una **lista con repetidos**.
- Añadir un nuevo elemento (`.add()`).
- **Eliminar un elemento** (`.remove()` o `.discard()`).
- Eliminar **todos los elementos** (`.clear()`).
- Añadir **varios elementos** a la vez (`.update()`).
- **Eliminar un elemento aleatorio** (`.pop()`).
- Eliminar elementos que cumplan una condición usando **bucle y copia**.
- **Copiar** un conjunto (`.copy()`).

B. Operaciones Lógicas

- Comprobar si un elemento **pertenece** al conjunto (`in`).
- **Unir** dos o más conjuntos (`.union()` o `|`).
- Mostrar la **intersección** (`.intersection()` o `&`).
- Mostrar la **diferencia** (`.difference()` o `-`).
- Mostrar la **diferencia simétrica** (`.symmetric_difference()` o `^`).
- Comprobar si un conjunto es **subconjunto** (`.issubset()`).
- Comprobar si un conjunto es **superconjunto** (`.issuperset()`).
- Comprobar si dos conjuntos **no tienen elementos en común** (`.isdisjoint()`).
- Verificar si un conjunto está **vacío** (`len() == 0`).
- Mantener solo los elementos comunes (`.intersection_update()`).
- Eliminar elementos presentes en otro conjunto (`.difference_update()`).
- Comprobar si dos conjuntos son **iguales**.

C. Análisis y Filtrado

- Contar cuántos elementos hay (`len()`).
- Calcular la **suma**.
- Calcular el valor **máximo/mínimo**.
- Mostrar solo los **números pares/impares**.
- Mostrar solo los números **mayores que 5**.
- Mostrar los números **menores o iguales a 3**.
- Mostrar los números **mayores que 3 y menores que 7**.
- Mostrar los números que sean **múltiplos de 2 o 3**.

- Mostrar los números **divisibles por 2 y 3 a la vez**.
- Mostrar las palabras que contienen la letra **“a”**.
- Mostrar las palabras que **empiezan con “m” o “p”**.
- Mostrar las palabras que tengan **más de 4 / 5 / exactamente 4 / exactamente 5 letras**.
- Mostrar las palabras que contengan **al menos una vocal**.

D. Transformación de Elementos

- Convertir una cadena en un **conjunto de caracteres**.
- Convertir un conjunto en una **lista** o una **tupla**.
- Convertir a **string separado por comas** (`.join()`).
- Crear conjunto con números **al cuadrado**.
- Crear conjunto con números **al cubo**.
- Crear conjunto con números **multiplicados por 2/3/5**.
- Crear conjunto con palabras en **mayúsculas/minúsculas**.
- Crear conjunto con palabras **invertidas**.

V. Ejercicios Python: Tuplas (1-100)

A. Creación y Acceso

- Crear una **tupla** con cinco números.
- Acceder al **primer y último elemento**.
- Mostrar elementos junto con sus **posiciones** (`enumerate()`).
- Crear tupla a partir de un **rango de números**.
- Obtener una **porción (slice)**.
- **Combinar dos tuplas** de igual longitud en una sola de pares (`zip()`).
- Crear tupla con los **cuadrados** de los números del 1 al 5.
- Crear tupla con los **caracteres** de una palabra.
- Crear tupla con los **valores absolutos**.
- **Desempaquetar** (`unpacking`) tres coordenadas.
- Mostrar el **segundo y cuarto color**.
- Mostrar los **tres primeros días**.
- Mostrar la **suma de los tres primeros/últimos** elementos.

B. Propiedades y Manipulación

- Contar cuántas veces aparece un elemento (`.count()`).
- Encontrar el **índice** de un elemento (`.index()`).
- Mostrar la **longitud** (`len()`).
- **Unir** tres tuplas (`+`).
- Calcular la **suma, promedio y máximo/mínimo**.
- Calcular el **producto** de todos los números.
- Encontrar la **mediana**.
- Verificar si **al menos un número es impar** (`any()`).
- Determinar si **todos los números son positivos** (`all()`).
- Mostrar la **cantidad total de caracteres** de las cadenas.

C. Conversiones y Filtrado

- Convertir a **lista**.
- Eliminar **duplicados** (via `set`).
- Tupla con el **doble de cada número**.
- Convertir una cadena con números separados por comas en una **tupla de enteros**.
- Mostrar los números **mayores que 5**.
- Mostrar los números **impares**.
- Ordenar **alfabéticamente**.
- Ordenar por **longitud** (`key=len`).
- Mostrar palabras que **empiezan con vocal**.
- Mostrar nombres que **empiezan con "M"**.
- Mostrar nombres que **terminan en "a"**.
- Mostrar los números que están **entre 10 y 50**.
- Mostrar solo los **múltiplos de 3**.
- Mostrar solo los **pares**.

VI. Ejercicios Python: Diccionarios (1-100)

A. Creación y Modificación

- Crear diccionarios de: **personas y edades**, **frutas y precios**, **estudiantes y calificaciones**, **ciudades y población**, **asignaturas y notas**, **productos y stock**, **países y capitales**, **empleados y salarios**, **coches y colores**, **artistas y géneros**.
- **Acceder** al valor por una clave.
- **Añadir** un nuevo par clave:valor.
- **Modificar** el valor de una clave existente.
- **Eliminar** un par clave:valor (`del`).

B. Búsqueda y Vistas

- Mostrar todas las **claves** (`.keys()`).
- Mostrar todos los **valores** (`.values()`).
- Mostrar todos los pares **clave-valor** (`.items()`).
- Comprobar si una **clave existe** (`in`).
- Obtener un valor usando **`.get()`** (para evitar errores).

VII. Ejercicios Python: Funciones (1-100)

A. Funciones Aritméticas y Lógicas

- Funciones de cálculo: **Suma, Resta, Multiplicación, División** de dos números.
- Calcular el **doble/triple** de un número.
- Devolver el número **mayor/menor** entre dos.
- Calcular **promedio** (dos o tres números).
- Devolver el **valor absoluto**.
- Devolver la **raíz cuadrada/cúbica**.

- Devolver el **valor negativo** de un número.
- Devolver **inverso** (1/n), manejo de división por cero.
- **Diferencia absoluta**.
- **Evaluar paridad**: ¿es par/impar?.
- **Evaluar signo**: ¿es positivo/negativo/cero?.
- **Evaluar divisibilidad**: ¿es divisible por 3/5/7/10?.
- ¿Es múltiplo de **2 y 3** / **2 y 5** / **3 y no 2**?.
- ¿Es múltiplo de **3 o 7** / **2 o 5**?.
- ¿Son **múltiplos uno del otro**?.
- ¿Es **mayor que 10** / **entre 0 y 100** / **entre 50 y 100**?.

B. Funciones de Listas y Colecciones

- Mostrar elementos de una **lista**.
- Devolver la **suma** de los elementos.
- Devolver la **longitud** de la lista.
- Devolver el elemento **mayor/menor**.
- Devolver el **primer/último/segundo/penúltimo** elemento.
- Devolver la lista **invertida**.
- Devolver solo los **números impares**.
- Contar el **número de elementos pares**.
- **Eliminar duplicados** (usando `set`).
- Devolver `True` si la lista está **vacía**.
- Devolver `True` si contiene el **número 0** / **al menos un número par** / **negativos**.
- Devolver `True` si **todos los elementos son pares** / **positivos y pares**.
- Devolver la lista con sus elementos **al cuadrado**.
- Devolver el **promedio** de sus elementos.
- Devolver la **suma de los positivos**.

C. Funciones de Cadenas de Texto

- Imprimir "Hola" y el **nombre**.
- Imprimir **nombre y apellido** juntos.
- Imprimir texto en **mayúsculas**.
- Devolver la **longitud** del texto.
- Devolver texto **invertido**.
- Devolver el **número de vocales**.
- Devolver el texto **sin espacios**.
- Devolver el **número de palabras**.
- Devolver `True` si **contiene la letra “a”** / **“o”** / **“a” y “e”**.
- Devolver la **primera letra**.
- Devolver `True` si tiene **más de 5 caracteres**.
- Devolver `True` si **empieza con “A”**.
- Devolver `True` si **empieza y termina con la misma letra**.
- Devolver `True` si contiene **solo letras**.
- Devolver la **cantidad de letras mayúsculas**.

VIII. Ejercicios Python: Programación Orientada a Objetos (OOP) (1-100)

A. Clases Base y Métodos Fundamentales

- **Clase `Persona`**: atributos nombre, edad, método `saludar()`.
- **Clase `Rectangulo`**: atributos base, altura, método `area()`.
- **Clase `Coche`**: atributos marca, modelo, método `descripcion()`.
- **Clase `CuentaBancaria`**: atributos titular, saldo, métodos `depositar()`, `retirar()`.
- **Clase `Animal`**: atributo nombre, método `hablar()`.
- **Clase `Libro`**: atributos titulo, autor, método `info()`.
- **Clase `Empleado`**: atributos nombre, sueldo, método `aumentar_sueldo()`.
- **Clase `Estudiante`**: atributos nombre, notas (lista), método `promedio()`.
- **Clase `Producto`**: atributos nombre, precio, método `aplicar_descuento()`.
- **Clase `Mascota`**: atributos nombre, tipo, método `descripcion()`.
- **Clase `Triangulo`**: atributos base, altura, método `area()`.
- **Clase `Vehiculo`**: atributos marca, modelo, método `detalle()`.
- **Clase `Alumno`**: atributos nombre, notas, método `aprobado()` (promedio ≥ 6).
- **Clase `Profesor`**: atributos nombre, asignatura, método `presentarse()`.
- **Clase `Circulo`**: atributo radio, método `area()`.

B. Herencia y Extensión de Clases

- **Herencia Animal:** `Perro` y `Gato` (sobrescriben `hablar()`); `PerroConRaza`; `PerroConEdad`; `PerroConColorYEdad` (cambiar color/edad); `PerroConVacunas`; `PerroConPeso`; `PerroConEnergia` (método `jugar()`).
- **Herencia CuentaBancaria:**
  - Subclases con **consultas/bonificaciones**: `CuentaBancariaSegura` (mostrar saldo), `CuentaBancariaVIP` (bono +100).
  - Subclases con **validación/cálculo**: `CuentaCorriente` (limite), `CuentaConInteres` (+5% interés), `CuentaConRetiroMinimo`, `CuentaAhorro` (cálculo de interés anual).
  - Subclases con **operaciones complejas**: `CuentaConTransferencia`, `CuentaConRetiroSeguro` (límite de 500).
  - Subclases con **estado**: `CuentaConBloqueo`.
  - Subclases con **validación de depósito**: `CuentaConDepositoCondicional`.
- **Herencia Empleado:** `EmpleadoTemporal` (meses contrato), `EmpleadoConBono` (+200 sueldo), `EmpleadoConAumento` (aumento con porcentaje), `EmpleadoConProyecto` (lista proyectos), `EmpleadoConHorasExtras` (sueldo total), `EmpleadoConDepartamento`.
- **Herencia Libro:** `LibroDigital` (formato), `LibroFisico` (páginas), `LibroConPrestamo`, `LibroConResenas`, `LibroConDisponibilidad`, `LibroConPaginasLeidas` (seguimiento de lectura).

- **Herencia Estudiante/Alumno:** `EstudianteConEmail`, `AlumnoConEdad`, `EstudianteConCurso`, `AlumnoConPromocion` (promedio ≥ 7), `EstudianteConCursoYPromedio` (promedio redondeado), `AlumnoConPromedioYCurso` (examen recuperación < 6), `AlumnoConNotasExtras` (método agregar nota), `AlumnoConRegistro` (diccionario de notas con fecha), `EstudianteConPromedioCondicional`.
- **Herencia Vehiculo/Coche/Bicicleta/Smartphone:** `Auto` (color), `VehiculoElectrico` (batería), `SmartphoneConAlmacenamiento`, `BicicletaConColor`, `VehiculoConPuertas`, `SmartphoneConCamara`, `BicicletaConMarchas`, `SmartphoneConApps` (instalar app), `BicicletaConCambio`, `VehiculoConVelocidad` (acelerar), `VehiculoConColor` (cambiar color).

IX. Ejercicios Python: Formato de Salida (1-100)

A. Formato Numérico y Alineación (F-string y .format())

- **Decimales**: Muestra número con 2 decimales (`:.2f`).
- **Relleno de ceros**: Relleno de ceros a la izquierda (`:0>5`, `:0>10`).
- **Signo**: Muestra signo positivo y negativo (`:+`).
- **Notación científica** (`:e`).
- **Porcentaje** (`:.2%`).
- **Separador de miles** (`,`).
- **Alineación a la derecha** (`:>10`).
- **Alineación a la izquierda** (`:<10`).
- **Centrado** (`:^10`).

B. Relleno y Combinaciones

- Relleno con **puntos**.
- Relleno con **asteriscos**.
- Relleno con **guiones**.
- Combinación de alineación, relleno y decimales.

C. Caracteres y Sintaxis de String

- Mostrar texto con **comillas** dentro de la cadena.
- Mostrar texto con **salto de línea** (`\n`).
- Mostrar texto con **tabulación** (`\t`).
- **Escapar** caracteres especiales (barra invertida `\\`).
- Usar **triple comilla** para texto multilínea.
- Insertar variables en texto usando **`str.format()`**.
- Ordenar los valores en `str.format()` según **índice**.

X. Ejercicios Python: Archivos, CSV, JSON y Juegos (1-100 en cada sección)

A. Manejo General de Archivos (Ficheros)

- **Crear y escribir** archivo (`open` modo "w").
- **Leer** y mostrar el contenido (`.read()`).
- **Añadir** una línea (`open` modo "a").
- **Listar** archivos del directorio (`os.listdir`).
- **Comprobar si existe** (`Path.exists`).
- **Crear/Eliminar carpetas** (`os.makedirs`, `os.rmdir`).
- **Mover/Copiar/Renombrar** archivos y carpetas (`shutil.move`, `shutil.copy`, `shutil.copytree`, `os.rename`).
- Obtener **tamaño** en bytes (`os.path.getsize`).
- Mostrar **ruta absoluta**.
- Listar archivos por patrón (`glob.glob`).
- **Compresión** (ZIP) y descompresión (`zipfile`, `shutil.make_archive`).
- **Escribir/leer archivos binarios**.
- **Contar líneas**.
- **Leer línea por línea** (`enumerate`).
- **Filtrar líneas** que contienen una palabra.
- **Sumar números** de un archivo de texto.
- Mostrar la **extensión** de los archivos.
- **Rotación de logs** por tamaño.
- Comparar si dos archivos son **iguales** (`filecmp.cmp`).
- **Archivos temporales** (`tempfile.NamedTemporaryFile`).
- **Serialización** (`pickle`, `shelve`).
- **Detección de duplicados** por hash.

B. Manejo Específico de Archivos CSV

- Crear archivo CSV **vacío**.
- Escribir filas con **`csv.writer()`**.
- Escribir varias filas con **`writerows()`**.
- Leer contenido con **`csv.reader()`**.
- Agregar filas (`open` modo "a").
- Usar **`csv.DictReader()`** (lista de diccionarios).
- Usar **`csv.DictWriter()`** (escribir con claves).
- **Contar filas/columnas**.
- **Sumar/promediar** valores de una columna.
- Mostrar **máximo/mínimo** de una columna.
- **Filtrar filas** por condición (edad > 25, precio > 1, etc.).
- **Ordenar datos** por columna numérica.

C. Manejo Específico de Archivos JSON

- **Guardar/Leer** diccionario simple (`json.dump()`, `json.load()`).
- Guardar/Leer **listas**.
- Guardar/Leer **lista de diccionarios** (empleados, productos, alumnos).
- Manejo de **datos anidados** (diccionarios dentro de diccionarios/listas).
- **Acceder a valores anidados** (ej. ciudad en dirección, nota de materia).
- **Filtrar** estructuras complejas (ej. empleados de Marketing, autos con año >= 2020).
- Calcular el **promedio** de una lista anidada (temperaturas).

D. Juegos (Lógica de Programación y Aleatoriedad)

- **Adivinar números** (con `random.randint`).
- **Pistas de proximidad**: "Muy cerca" (≤ 5/3/2), "Caliente", "Cálido", "Frío".
- **Límite de intentos** y conteo regresivo.
- **Puntuación** con decremento por fallo.
- **Adivinar colores**.
- **Pistas de color** (primera letra, longitud, vocales, consonantes).
- **Piedra, Papel o Tijera**.
- **Dados** (suma, par/impar, mayor/menor que 7/8, lanzamiento repetitivo).
- **Moneda** (cara o cruz, con repetición).

Ejercicios python variables y operadores

---

1. **Enunciado: Pedir al usuario dos números y mostrar su suma.**

Código:

```
# 1 - Suma de dos números
a = float(input("Introduce el primer número: "))
b = float(input("Introduce el segundo número: "))
suma = a + b
print(f"La suma de {a} y {b} es {suma}")
```

Explicación: Lee dos valores (como `float` para aceptar decimales), los suma y muestra el resultado con formato.

---

2. **Enunciado: Pedir un número y mostrar si es par o impar.**

Código:

```
# 2 - Par o impar
n = int(input("Introduce un número entero: "))
if n % 2 == 0:
    print(f"{n} es par")
else:
    print(f"{n} es impar")
```

Explicación: Usa el operador módulo % para comprobar el resto al dividir por 2; si es 0 es par, si no, impar.

---

3. **Enunciado: Calcular el factorial de un número no negativo (con `for`).**

Código:

```
# 3 - Factorial (for)
n = int(input("Introduce un entero no negativo: "))
if n < 0:
    print("El factorial no está definido para números negativos.")
else:
    fact = 1
    for i in range(2, n+1):
        fact *= i
    print(f"{n}! = {fact}")
```

Explicación: Itera desde 2 hasta `n` multiplicando acumuladamente para obtener el factorial; manejo simple de caso negativo.

4.

Enunciado: Calcular la suma de los dígitos de un número entero (positivo).
Código:

```
# 4 - Suma de dígitos
n = int(input("Introduce un número entero positivo: "))
n = abs(n)
s = 0
while n > 0:
    s += n % 10
    n //= 10
print("La suma de los dígitos es", s)
```

Explicación: Extrae dígito a dígito con `% 10` y reduce el número con división entera `//`.

5.

Enunciado: Mostrar la tabla de multiplicar de un número (del 1 al 10).
Código:

```
# 5 - Tabla de multiplicar
n = int(input("Introduce un número para ver su tabla: "))
for i in range(1, 11):
    print(f"{n} x {i} = {n * i}")
```

Explicación: Bucle `for` de 1 a 10 para mostrar los productos.

6.

Enunciado: Comprobar si un número es primo (método sencillo).
Código:

```
# 6 - Comprobar primo
n = int(input("Introduce un entero mayor que 1: "))
if n <= 1:
    print("Debe introducir un número mayor que 1.")
else:
    es_primo = True
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            es_primo = False
            break
    print(f"{n} {'es primo' if es_primo else 'no es primo'}")
```

Explicación: Prueba divisores hasta la raíz cuadrada; si encuentra divisor, no es primo.

7.

Enunciado: Invertir un número entero (ej. 123 -> 321).

Código:

```
# 7 - Invertir número
n = int(input("Introduce un número entero: "))
signo = -1 if n < 0 else 1
n = abs(n)
inv = 0
while n > 0:
    inv = inv * 10 + n % 10
    n //= 10
inv *= signo
print("Número invertido:", inv)
```

Explicación: Construye el número invertido acumulando dígitos; conserva el signo original.

8.

Enunciado: Contar cuántos dígitos tiene un número entero.

Código:

```
# 8 - Contar dígitos
n = int(input("Introduce un número entero: "))
n = abs(n)
if n == 0:
    print("El número tiene 1 dígito")
else:
    contador = 0
    while n > 0:
        contador += 1
        n //= 10
    print(f"El número tiene {contador} dígitos")
```

Explicación: Divide repetidamente por 10 y cuenta iteraciones; caso especial para 0.

9.

Enunciado: Calcular el máximo y mínimo entre tres números.

Código:

```
# 9 - Máximo y mínimo de tres números
a = float(input("a: "))
b = float(input("b: "))
c = float(input("c: "))
maximo = max(a, b, c)
minimo = min(a, b, c)
print(f"Máximo: {maximo}")
print(f"Mínimo: {minimo}")
```

Explicación: Usa las funciones incorporadas `max` y `min` para simplicidad.

10.

Enunciado: Calcular la media (promedio) de N números introducidos por el usuario.
Código:

```
# 10 - Media de N números
n = int(input("¿Cuántos números vas a introducir? "))
if n <= 0:
    print("Cantidad inválida.")
else:
    suma = 0.0
    for i in range(n):
        x = float(input(f"Número {i+1}: "))
        suma += x
    media = suma / n
    print(f"La media es {media}")
```

Explicación: Suma los valores y divide por la cantidad para obtener la media; valida n.

11.

Enunciado: Determinar si un número es capicúa (palíndromo numérico).
Código:

```
# 11 - Número capicúa
n = input("Introduce un número entero: ").strip()
# Permitimos signo inicial
if n.startswith('-'):
    core = n[1:]
else:
    core = n
if core == core[::-1]:
    print(f"{n} es capicúa")
else:
    print(f"{n} no es capicúa")
```

Explicación: Trabaja con la representación de cadena; compara con su reverso ignorando el signo.

12.

Enunciado: Mostrar los divisores de un número.

Código:

```
# 12 - Divisores de un número
n = int(input("Introduce un número entero positivo: "))
if n <= 0:
    print("Introduce un número positivo.")
else:
    divisores = []
    for i in range(1, int(n**0.5)+1):
        if n % i == 0:
            divisores.append(i)
            if i != n // i:
                divisores.append(n // i)
    divisores.sort()
    print("Divisores:", divisores)
```

Explicación: Busca divisores hasta la raíz para eficiencia, añade el par complementario y ordena la lista.

13.

Enunciado: Comprobar si dos números son múltiplos entre sí.

Código:

```
# 13 - Comprobar múltiplos
a = int(input("Introduce el primer entero: "))
b = int(input("Introduce el segundo entero: "))
if a == 0 or b == 0:
    print("Si uno es 0, solo es múltiplo si el otro también es 0.")
else:
    if a % b == 0:
        print(f"{a} es múltiplo de {b}")
    elif b % a == 0:
        print(f"{b} es múltiplo de {a}")
    else:
        print("No son múltiplos entre sí")
```

Explicación: Usa % para ver si uno divide exactamente al otro; trata el caso con ceros.

14.

Enunciado: Calcular el MCD (máximo común divisor) de dos números usando el algoritmo de Euclides.

Código:

```
# 14 - MCD con Euclides
a = abs(int(input("Introduce el primer entero: ")))
b = abs(int(input("Introduce el segundo entero: ")))
while b != 0:
    a, b = b, a % b
print("MCD:", a)
```

Explicación: Implementa el algoritmo eficiente de Euclides intercambiando y tomando restos hasta que el segundo sea 0.

15.

Enunciado: Calcular el MCM (mínimo común múltiplo) de dos números.

Código:

```
# 15 - MCM usando MCD
def mcd(x, y):
    x, y = abs(x), abs(y)
    while y:
        x, y = y, x % y
    return x

a = int(input("a: "))
b = int(input("b: "))
if a == 0 or b == 0:
    print("El MCM con 0 es 0.")
else:
    resultado = abs(a * b) // mcd(a, b)
    print("MCM:", resultado)
```

Explicación: Calcula MCD y usa la relación `mcm = |a*b| / mcd`; maneja ceros.

16.

Enunciado: Generar y mostrar los N primeros números pares.

Código:

```
# 16 - N primeros pares
n = int(input("¿Cuántos números pares quieres? "))
count = 0
num = 0
pares = []
while count < n:
    pares.append(num)
    num += 2
    count += 1
print("Pares:", pares)
```

Explicación: Empieza en 0 y suma 2 en cada iteración para obtener pares consecutivos.

17.

Enunciado: Generar y mostrar los N primeros números impares.

Código:

```
# 17 - N primeros impares
n = int(input("¿Cuántos números impares quieres? "))
impares = [2*i + 1 for i in range(n)]
print("Impares:", impares)
```

Explicación: Usa comprensión de listas con fórmula `2*i+1` para impares positivos.

18.

Enunciado: Calcular la suma de los números pares entre 1 y N (inclusive).

Código:

```
# 18 - Suma de pares hasta N
n = int(input("Introduce N (entero positivo): "))
suma = 0
for i in range(2, n+1, 2):
    suma += i
print(f"Suma de pares hasta {n}: {suma}")
```

Explicación: Recorre de 2 a `n` con paso 2 y acumula; eficiente y legible.

19.

Enunciado: Calcular la potencia `a^b` con `b` entero no negativo (sin usar `**`).

Código:

```
# 19 - Potencia por multiplicación
a = float(input("Base a: "))
b = int(input("Exponente b (entero no negativo): "))
if b < 0:
    print("Exponente negativo no permitido en este ejercicio.")
else:
    res = 1.0
    for _ in range(b):
        res *= a
    print(f"{a}^{b} = {res}")
```

Explicación: Multiplica la base `b` veces; evita usar operador `**` para practicar bucles.

20.

Enunciado: Redondear un número a su parte entera (truncamiento hacia cero) sin usar `int()` directamente sobre el número.

Código:

```
# 20 - Truncar hacia cero
x = float(input("Introduce un número real: "))
if x >= 0:
    entero = x // 1
else:
    entero = -((-x) // 1)
# Convertimos a int para mostrar sin .0
entero = int(entero)
print(f"Parte entera (truncada hacia cero) de {x} es {entero}")
```

Explicación: Usa división entera `//` y manipulación de signo para conseguir truncamiento hacia cero consistente para positivos y negativos; finalmente convierte a `int` para presentación.

21.

Enunciado: Calcular la diferencia absoluta entre dos números.

Código:

```
# 21 - Diferencia absoluta
a = float(input("Primer número: "))
b = float(input("Segundo número: "))
diferencia = abs(a - b)
print("La diferencia absoluta es:", diferencia)
```

Explicación: Usa la función `abs()` para obtener el valor absoluto de la diferencia sin importar cuál sea mayor.

22.

Enunciado: Calcular el cuadrado y el cubo de un número.

Código:

```
# 22 - Cuadrado y cubo
n = float(input("Introduce un número: "))
print(f"Cuadrado: {n**2}")
print(f"Cubo: {n**3}")
```

Explicación: Eleva el número al cuadrado (`n**2`) y al cubo (`n**3`) usando el operador de potencia.

23.

Enunciado: Verificar si un número es positivo, negativo o cero.
Código:

```
# 23 - Signo del número
n = float(input("Introduce un número: "))
if n > 0:
    print("El número es positivo")
elif n < 0:
    print("El número es negativo")
else:
    print("El número es cero")
```

Explicación: Usa condicionales para determinar el signo del número.

24.

Enunciado: Convertir grados Celsius a Fahrenheit.
Código:

```
# 24 - Celsius a Fahrenheit
c = float(input("Temperatura en Celsius: "))
f = (c * 9/5) + 32
print(f"{c}°C equivalen a {f}°F")
```

Explicación: Aplica la fórmula de conversión estándar `F = C * 9/5 + 32`.

25.

Enunciado: Convertir grados Fahrenheit a Celsius.
Código:

```
# 25 - Fahrenheit a Celsius
f = float(input("Temperatura en Fahrenheit: "))
c = (f - 32) * 5/9
print(f"{f}°F equivalen a {c}°C")
```

Explicación: Aplica la fórmula inversa `C = (F - 32) * 5/9`.

26.

Enunciado: Calcular el área de un círculo dado su radio.
Código:

```
# 26 - Área de un círculo
import math
r = float(input("Radio del círculo: "))
area = math.pi * r**2
print(f"Área del círculo: {area}")
```

Explicación: Usa la constante `math.pi` y la fórmula del área πr^2.

27.

Enunciado: Calcular el perímetro de un círculo dado su radio.

Código:

```
# 27 - Perímetro de un círculo
import math
r = float(input("Radio del círculo: "))
perimetro = 2 * math.pi * r
print(f"Perímetro del círculo: {perimetro}")
```

Explicación: Utiliza la fórmula $2\pi r$ para calcular el perímetro.

28.

Enunciado: Calcular el área de un triángulo dados base y altura.

Código:

```
# 28 - Área de triángulo
base = float(input("Base del triángulo: "))
altura = float(input("Altura del triángulo: "))
area = (base * altura) / 2
print(f"Área del triángulo: {area}")
```

Explicación: Usa la fórmula básica del triángulo $(base \times altura) / 2$.

29.

Enunciado: Calcular el área de un cuadrado dado su lado.

Código:

```
# 29 - Área de un cuadrado
lado = float(input("Introduce el lado: "))
area = lado * lado
print(f"Área del cuadrado: {area}")
```

Explicación: Multiplica el lado por sí mismo para obtener el área.

30.

Enunciado: Calcular el área de un rectángulo.

Código:

```
# 30 - Área de rectángulo
base = float(input("Base: "))
altura = float(input("Altura: "))
area = base * altura
print(f"Área del rectángulo: {area}")
```

Explicación: Multiplica la base por la altura; ambas deben ser positivas.

31.

Enunciado: Calcular el volumen de un cubo dado su arista.

Código:

```
# 31 - Volumen de un cubo
arista = float(input("Arista del cubo: "))
volumen = arista ** 3
print(f"Volumen del cubo: {volumen}")
```

Explicación: Eleva la arista al cubo (`arista³`) para hallar el volumen.

32.

Enunciado: Calcular el perímetro de un rectángulo.

Código:

```
# 32 - Perímetro de rectángulo
base = float(input("Base: "))
altura = float(input("Altura: "))
perimetro = 2 * (base + altura)
print(f"Perímetro del rectángulo: {perimetro}")
```

Explicación: Suma las longitudes de los lados con `2 × (base + altura)`.

33.

Enunciado: Calcular el promedio de tres números.

Código:

```
# 33 - Promedio de tres números
a = float(input("Número 1: "))
b = float(input("Número 2: "))
c = float(input("Número 3: "))
media = (a + b + c) / 3
print(f"Promedio: {media}")
```

Explicación: Suma los tres números y divide el total entre tres.

34.

Enunciado: Comprobar si un número es divisible entre 5.

Código:

```
# 34 - Divisible entre 5
n = int(input("Introduce un número: "))
if n % 5 == 0:
    print(f"{n} es divisible entre 5")
else:
    print(f"{n} no es divisible entre 5")
```

Explicación: Usa el operador % para comprobar si el resto de dividir por 5 es cero.

35.

Enunciado: Mostrar todos los múltiplos de 3 entre 1 y 50.

Código:

```
# 35 - Múltiplos de 3
for i in range(3, 51, 3):
    print(i, end=" ")
```

Explicación: Recorre los múltiplos de 3 desde 3 hasta 50 con un paso de 3 en el rango.

36.

Enunciado: Calcular el cuadrado de los números del 1 al 10.

Código:

```
# 36 - Cuadrados del 1 al 10
for i in range(1, 11):
    print(f"{i}^2 = {i**2}")
```

Explicación: Usa un bucle `for` para imprimir los cuadrados de cada número.

37.

Enunciado: Calcular la suma de los números del 1 al N.

Código:

```
# 37 - Suma de 1 a N
n = int(input("Introduce N: "))
suma = 0
for i in range(1, n + 1):
    suma += i
print(f"La suma de 1 a {n} es {suma}")
```

Explicación: Acumula con un bucle desde 1 hasta N inclusive.

38.

Enunciado: Calcular la suma de los números impares entre 1 y N.

Código:

```
# 38 - Suma de impares hasta N
n = int(input("Introduce N: "))
suma = 0
for i in range(1, n + 1, 2):
    suma += i
print(f"Suma de impares hasta {n}: {suma}")
```

Explicación: Recorre con paso de 2 desde 1 y acumula los valores.

39.

Enunciado: Calcular el doble y el triple de un número.
Código:

```
# 39 - Doble y triple
n = float(input("Introduce un número: "))
print(f"Doble: {2*n}")
print(f"Triple: {3*n}")
```

Explicación: Multiplica el número por 2 y 3 respectivamente.

40.

Enunciado: Verificar si un número es divisible entre 2 y entre 3 al mismo tiempo.
Código:

```
# 40 - Divisible entre 2 y 3
n = int(input("Introduce un número entero: "))
if n % 2 == 0 and n % 3 == 0:
    print(f"{n} es divisible entre 2 y 3")
else:
    print(f"{n} no es divisible entre ambos")
```

Explicación: Usa una condición doble (`and`) para verificar la divisibilidad simultánea.

41.

Enunciado: Mostrar los números del 1 al 20.
Código:

```
# 41 - Mostrar números del 1 al 20
for i in range(1, 21):
    print(i, end=" ")
```

Explicación: Usa un bucle `for` con `range(1,21)` para recorrer los números del 1 al 20 inclusive.

42.

Enunciado: Mostrar los números del 20 al 1 en orden descendente.
Código:

```
# 42 - Números del 20 al 1
for i in range(20, 0, -1):
    print(i, end=" ")
```

Explicación: Usa `range(20, 0, -1)` para contar hacia atrás desde 20 hasta 1.

43.

Enunciado: Calcular la suma de los números pares entre 1 y 100.
Código:

```
# 43 - Suma de pares del 1 al 100
suma = 0
for i in range(2, 101, 2):
    suma += i
print("Suma de pares:", suma)
```

Explicación: Recorre solo los números pares con paso 2 y acumula sus valores.

44.

Enunciado: Calcular la suma de los números impares entre 1 y 100.
Código:

```
# 44 - Suma de impares del 1 al 100
suma = 0
for i in range(1, 101, 2):
    suma += i
print("Suma de impares:", suma)
```

Explicación: Igual que el anterior, pero empezando en 1 con paso 2.

45.

Enunciado: Mostrar los números que son múltiplos de 4 entre 1 y 50.
Código:

```
# 45 - Múltiplos de 4
for i in range(4, 51, 4):
    print(i, end=" ")
```

Explicación: Inicia en 4 y usa paso 4 para mostrar los múltiplos de cuatro.

46.

Enunciado: Contar cuántos números del 1 al 100 son divisibles entre 5.
Código:

```
# 46 - Contar múltiplos de 5
contador = 0
for i in range(1, 101):
    if i % 5 == 0:
        contador += 1
print("Hay", contador, "números divisibles entre 5 entre 1 y 100.")
```

Explicación: Recorre los números del 1 al 100 y aumenta el contador cada vez que el número es múltiplo de 5.

47.

Enunciado: Calcular el promedio de los números del 1 al 10.
Código:

```
# 47 - Promedio de 1 a 10
suma = 0
for i in range(1, 11):
    suma += i
media = suma / 10
print("Promedio:", media)
```

Explicación: La media se calcula sumando del 1 al 10 y dividiendo entre 10.

48.

Enunciado: Mostrar los números comprendidos entre dos valores dados por el usuario.
Código:

```
# 48 - Números entre A y B
a = int(input("Primer número: "))
b = int(input("Segundo número: "))
if a < b:
    for i in range(a, b + 1):
        print(i, end=" ")
else:
    for i in range(a, b - 1, -1):
        print(i, end=" ")
```

Explicación: Permite mostrar los números ascendentes o descendentes según el orden de entrada.

49.

Enunciado: Calcular el promedio de N números introducidos por el usuario hasta ingresar 0.
Código:

```
# 49 - Promedio hasta ingresar 0
suma = 0
contador = 0
while True:
    n = float(input("Introduce un número (0 para terminar): "))
    if n == 0:
        break
    suma += n
    contador += 1
if contador > 0:
    print("Promedio:", suma / contador)
else:
    print("No se introdujeron números.")
```

Explicación: Usa un bucle infinito que se detiene al ingresar 0, acumulando la suma y el contador.

50.

Enunciado: Calcular el cuadrado de un número sin usar el operador `**`.

Código:

```
# 50 - Cuadrado sin usar **
n = float(input("Introduce un número: "))
cuadrado = n * n
print("Cuadrado:", cuadrado)
```

Explicación: Multiplica el número por sí mismo para calcular su cuadrado.

51.

Enunciado: Determinar si un número termina en 7.

Código:

```
# 51 - Termina en 7
n = int(input("Introduce un número: "))
if abs(n) % 10 == 7:
    print("El número termina en 7")
else:
    print("El número no termina en 7")
```

Explicación: El módulo 10 da el último dígito del número; se usa `abs()` para ignorar el signo.

52.

Enunciado: Mostrar la mitad de un número.

Código:

```
# 52 - Mitad de un número
n = float(input("Introduce un número: "))
print("La mitad es:", n / 2)
```

Explicación: Divide el número entre dos para mostrar su mitad.

53.

Enunciado: Mostrar el triple del cuadrado de un número.

Código:

```
# 53 - Triple del cuadrado
n = float(input("Introduce un número: "))
resultado = 3 * (n ** 2)
print("Triple del cuadrado:", resultado)
```

Explicación: Eleva al cuadrado y multiplica por tres.

54.

Enunciado: Determinar si un número tiene tres dígitos.

Código:

```
# 54 - Tiene tres dígitos
n = abs(int(input("Introduce un número: ")))
if 100 <= n <= 999:
    print("Tiene tres dígitos")
else:
    print("No tiene tres dígitos")
```

Explicación: Comprueba si el valor absoluto está en el rango [100, 999].

55.

Enunciado: Comprobar si un número es divisible entre 7.

Código:

```
# 55 - Divisible entre 7
n = int(input("Introduce un número: "))
if n % 7 == 0:
    print(f"{n} es divisible entre 7")
else:
    print(f"{n} no es divisible entre 7")
```

Explicación: Usa % para ver si el resto de la división entre 7 es cero.

56.

Enunciado: Mostrar los 10 primeros múltiplos de un número.

Código:

```
# 56 - 10 primeros múltiplos
n = int(input("Introduce un número: "))
for i in range(1, 11):
    print(f"{n} x {i} = {n * i}")
```

Explicación: Genera los primeros 10 múltiplos multiplicando por los valores del 1 al 10.

57.

Enunciado: Calcular el resto de una división sin usar el operador %.

Código:

```
# 57 - Resto sin usar %
a = int(input("Dividendo: "))
b = int(input("Divisor: "))
resto = a - (a // b) * b
print("Resto:", resto)
```

Explicación: Usa la relación `resto = dividendo - (cociente * divisor)` con división entera `//`.

58.

Enunciado: Calcular la potencia de 2 más cercana a un número dado.

Código:

```
# 58 - Potencia de 2 más cercana
import math
n = float(input("Introduce un número positivo: "))
exponente = round(math.log2(n))
resultado = 2 ** exponente
print(f"La potencia de 2 más cercana a {n} es {resultado}")
```

Explicación: Usa logaritmos en base 2 (`math.log2`) y redondea al exponente más cercano.

59.

Enunciado: Verificar si un número es múltiplo de otro.

Código:

```
# 59 - Múltiplo de otro
a = int(input("Primer número: "))
b = int(input("Segundo número: "))
if b != 0 and a % b == 0:
    print(f"{a} es múltiplo de {b}")
else:
    print(f"{a} no es múltiplo de {b}")
```

Explicación: Comprueba que el divisor no sea cero y que el resto sea cero.

60.

Enunciado: Calcular la suma de los cuadrados de los números del 1 al N.
Código:

```
# 60 - Suma de cuadrados
n = int(input("Introduce N: "))
suma = 0
for i in range(1, n + 1):
    suma += i ** 2
print(f"La suma de cuadrados hasta {n} es {suma}")
```

Explicación: Recorre todos los números del 1 al N y acumula sus cuadrados.

61.

Enunciado: Mostrar los números pares entre dos valores dados por el usuario.
Código:

```
# 61 - Pares entre dos números
a = int(input("Primer número: "))
b = int(input("Segundo número: "))
if a > b:
    a, b = b, a
for i in range(a, b + 1):
    if i % 2 == 0:
        print(i, end=" ")
```

Explicación: Ordena los valores si es necesario y recorre el rango, mostrando solo los números pares.

62.

Enunciado: Calcular la suma de los números comprendidos entre dos enteros.
Código:

```
# 62 - Suma entre dos números
a = int(input("Primer número: "))
b = int(input("Segundo número: "))
if a > b:
    a, b = b, a
suma = 0
for i in range(a, b + 1):
    suma += i
print(f"Suma entre {a} y {b}: {suma}")
```

Explicación: Si el primer número es mayor, se intercambian. Luego se suman todos los valores del rango.

63.

Enunciado: Mostrar los números impares entre 1 y 30.

Código:

```
# 63 - Impares del 1 al 30
for i in range(1, 31, 2):
    print(i, end=" ")
```

Explicación: Empieza en 1 y usa paso 2 en `range()` para listar los impares.

64.

Enunciado: Calcular el cubo de los números del 1 al 10.

Código:

```
# 64 - Cubos del 1 al 10
for i in range(1, 11):
    print(f"{i}^3 = {i**3}")
```

Explicación: Usa un bucle y el operador `**` para calcular el cubo de cada número.

65.

Enunciado: Calcular la suma de los cubos de los números del 1 al N.

Código:

```
# 65 - Suma de cubos
n = int(input("Introduce N: "))
suma = 0
for i in range(1, n + 1):
    suma += i**3
print(f"Suma de los cubos hasta {n}: {suma}")
```

Explicación: Itera de 1 a N sumando los cubos individuales.

66.

Enunciado: Calcular el promedio de los números pares entre 1 y N.

Código:

```
# 66 - Promedio de pares
n = int(input("Introduce N: "))
suma = 0
contador = 0
for i in range(2, n + 1, 2):
    suma += i
    contador += 1
if contador > 0:
    print("Promedio de pares:", suma / contador)
else:
    print("No hay números pares en el rango.")
```

Explicación: Acumula pares y divide entre su cantidad si existe al menos uno.

67.

Enunciado: Mostrar los números del 100 al 200 que sean divisibles entre 7.
Código:

```
# 67 - Divisibles entre 7 del 100 al 200
for i in range(100, 201):
    if i % 7 == 0:
        print(i, end=" ")
```

Explicación: Usa condicional `% 7 == 0` dentro de un bucle para filtrar los múltiplos.

68.

Enunciado: Calcular el producto de los números del 1 al N.
Código:

```
# 68 - Producto de 1 a N
n = int(input("Introduce N: "))
producto = 1
for i in range(1, n + 1):
    producto *= i
print("Producto:", producto)
```

Explicación: Multiplica cada número de 1 a N acumulando el resultado.

69.

Enunciado: Mostrar los números comprendidos entre 50 y 100 que sean múltiplos de 9.
Código:

```
# 69 - Múltiplos de 9 del 50 al 100
for i in range(50, 101):
    if i % 9 == 0:
        print(i, end=" ")
```

Explicación: Recorre el rango y usa el módulo para filtrar los múltiplos de 9.

70.

Enunciado: Mostrar la suma de los múltiplos de 5 menores que 50.
Código:

```
# 70 - Suma de múltiplos de 5
suma = 0
for i in range(5, 50, 5):
    suma += i
print("Suma:", suma)
```

Explicación: Recorre los múltiplos de 5 hasta 49 sumándolos con un paso de 5.

71.

Enunciado: Mostrar el cuadrado de los números pares del 1 al 20.
Código:

```
# 71 - Cuadrado de pares del 1 al 20
for i in range(2, 21, 2):
    print(f"{i}^2 = {i**2}")
```

Explicación: Recorre pares con paso 2 y eleva al cuadrado.

72.

Enunciado: Mostrar el cubo de los números impares del 1 al 15.
Código:

```
# 72 - Cubos de impares
for i in range(1, 16, 2):
    print(f"{i}^3 = {i**3}")
```

Explicación: Empieza en 1, avanza de 2 en 2 y calcula cubos de impares.

73.

Enunciado: Calcular la suma de todos los números múltiplos de 3 y 5 entre 1 y 100.
Código:

```
# 73 - Suma múltiplos de 3 y 5
suma = 0
for i in range(1, 101):
    if i % 3 == 0 and i % 5 == 0:
        suma += i
print("Suma:", suma)
```

Explicación: Suma solo los que son divisibles por 3 y 5 a la vez.

74.

Enunciado: Calcular la suma de los números pares y la suma de los impares entre 1 y N.
Código:

```
# 74 - Suma de pares e impares
n = int(input("Introduce N: "))
suma_pares = 0
suma_impares = 0
for i in range(1, n + 1):
    if i % 2 == 0:
        suma_pares += i
    else:
        suma_impares += i
print("Suma pares:", suma_pares)
print("Suma impares:", suma_impares)
```

Explicación: Usa una condición para separar pares e impares y sumar ambos por separado.

75.

Enunciado: Mostrar los números que son divisibles entre 3 pero no entre 5 entre 1 y 50.
Código:

```
# 75 - Divisibles entre 3 pero no 5
for i in range(1, 51):
    if i % 3 == 0 and i % 5 != 0:
        print(i, end=" ")
```

Explicación: Combina condiciones lógicas para filtrar los valores correctos.

76.

Enunciado: Calcular el número de dígitos de un número sin usar cadenas.
Código:

```
# 76 - Contar dígitos sin cadenas
n = abs(int(input("Introduce un número: ")))
if n == 0:
    print("El número tiene 1 dígito.")
else:
    contador = 0
    while n > 0:
        contador += 1
        n //= 10
    print("Tiene", contador, "dígitos.")
```

Explicación: Usa división entera sucesiva por 10 y cuenta los pasos.

77.

Enunciado: Comprobar si un número es múltiplo de 4 o de 6.
Código:

```
# 77 - Múltiplo de 4 o de 6
n = int(input("Introduce un número: "))
if n % 4 == 0 or n % 6 == 0:
    print("Es múltiplo de 4 o de 6")
else:
    print("No es múltiplo de 4 ni de 6")
```

Explicación: Usa el operador lógico `or` para aceptar cualquiera de las dos condiciones.

78.

Enunciado: Mostrar los números entre 1 y 100 que sean divisibles por 4 y por 6.
Código:

```
# 78 - Divisibles por 4 y 6
for i in range(1, 101):
    if i % 4 == 0 and i % 6 == 0:
        print(i, end=" ")
```

Explicación: Combina ambas condiciones con `and` para obtener múltiplos comunes.

79.

Enunciado: Calcular la suma de los números múltiplos de 10 entre 10 y 100.

Código:

```
# 79 - Suma de múltiplos de 10
suma = 0
for i in range(10, 101, 10):
    suma += i
print("Suma:", suma)
```

Explicación: Usa `range(10,101,10)` para recorrer los múltiplos y acumularlos.

80.

Enunciado: Calcular el producto de los números pares del 1 al 10.

Código:

```
# 80 - Producto de pares
producto = 1
for i in range(2, 11, 2):
    producto *= i
print("Producto de pares:", producto)
```

Explicación: Multiplica los números pares del 2 al 10 acumulando el resultado en una variable.

81.

Enunciado: Mostrar los números del 1 al N en orden descendente.

Código:

```
# 81 - Números descendentes hasta N
n = int(input("Introduce N: "))
for i in range(n, 0, -1):
    print(i, end=" ")
```

Explicación: Usa `range(n, 0, -1)` para contar hacia atrás desde N hasta 1.

82.

Enunciado: Calcular el factorial de un número usando `while`.

Código:

```
# 82 - Factorial con while
n = int(input("Introduce un número no negativo: "))
if n < 0:
    print("Número inválido")
else:
    fact = 1
    i = 2
    while i <= n:
        fact *= i
        i += 1
    print(f"{n}! = {fact}")
```

Explicación: Multiplica iterativamente desde 2 hasta n usando un bucle `while`.

83.

Enunciado: Calcular el número de cifras pares en un número entero.

Código:

```
# 83 - Contar cifras pares
n = abs(int(input("Introduce un número: ")))
contador = 0
while n > 0:
    if (n % 10) % 2 == 0:
        contador += 1
    n //= 10
print("Cantidad de cifras pares:", contador)
```

Explicación: Recorre los dígitos y cuenta solo los pares usando módulo 2.

84.

Enunciado: Mostrar los números múltiplos de 3 entre 1 y 30.

Código:

```
# 84 - Múltiplos de 3
for i in range(3, 31, 3):
    print(i, end=" ")
```

Explicación: Usa `range(3,31,3)` para listar los múltiplos de 3 consecutivos.

85.

Enunciado: Mostrar los números entre 1 y 50 que no sean múltiplos de 2 ni de 3.
Código:

```
# 85 - No múltiplos de 2 ni 3
for i in range(1, 51):
    if i % 2 != 0 and i % 3 != 0:
        print(i, end=" ")
```

Explicación: Filtra números impares que tampoco sean múltiplos de 3.

86.

Enunciado: Calcular la suma de los números de 1 a N que sean múltiplos de 4.
Código:

```
# 86 - Suma múltiplos de 4
n = int(input("Introduce N: "))
suma = 0
for i in range(4, n+1, 4):
    suma += i
print("Suma de múltiplos de 4:", suma)
```

Explicación: Recorre los múltiplos de 4 hasta N y los acumula.

87.

Enunciado: Calcular la suma de los dígitos de un número.
Código:

```
# 87 - Suma de dígitos
n = abs(int(input("Introduce un número: ")))
suma = 0
while n > 0:
    suma += n % 10
    n //= 10
print("Suma de dígitos:", suma)
```

Explicación: Extrae cada dígito con `%10` y suma, luego divide `//10` para recorrer todos.

88.

Enunciado: Mostrar el número más grande entre tres números dados.
Código:

```
# 88 - Máximo de tres números
a = float(input("a: "))
b = float(input("b: "))
c = float(input("c: "))
maximo = max(a, b, c)
print("Máximo:", maximo)
```

Explicación: Usa la función `max()` para obtener el mayor de tres valores.

89.

Enunciado: Mostrar el número más pequeño entre tres números dados.
Código:

```
# 89 - Mínimo de tres números
a = float(input("a: "))
b = float(input("b: "))
c = float(input("c: "))
minimo = min(a, b, c)
print("Mínimo:", minimo)
```

Explicación: Usa la función `min()` para obtener el menor de tres valores.

90.

Enunciado: Verificar si un número es divisible entre 2, 3 o 5.
Código:

```
# 90 - Divisible entre 2, 3 o 5
n = int(input("Introduce un número: "))
if n % 2 == 0 or n % 3 == 0 or n % 5 == 0:
    print(f"{n} es divisible entre 2, 3 o 5")
else:
    print(f"{n} no es divisible entre 2, 3 ni 5")
```

Explicación: Comprueba varias condiciones de divisibilidad usando `or`.

91.

Enunciado: Mostrar los números del 1 al N y su cuadrado.
Código:

```
# 91 - Números y sus cuadrados
n = int(input("Introduce N: "))
for i in range(1, n+1):
    print(f"{i}^2 = {i**2}")
```

Explicación: Recorre del 1 al N y calcula cada cuadrado.

92.

Enunciado: Mostrar los números del 1 al N y su cubo.

Código:

```
# 92 - Números y sus cubos
n = int(input("Introduce N: "))
for i in range(1, n+1):
    print(f"{i}^3 = {i**3}")
```

Explicación: Recorre del 1 al N y calcula cada cubo.

93.

Enunciado: Mostrar los múltiplos de 7 entre 1 y 100.

Código:

```
# 93 - Múltiplos de 7
for i in range(7, 101, 7):
    print(i, end=" ")
```

Explicación: Usa `range()` con paso 7 para mostrar todos los múltiplos de 7.

94.

Enunciado: Mostrar los números del 1 al 20 que sean pares y múltiplos de 3.

Código:

```
# 94 - Pares y múltiplos de 3
for i in range(1, 21):
    if i % 2 == 0 and i % 3 == 0:
        print(i, end=" ")
```

Explicación: Combina condiciones `and` para filtrar los números deseados.

95.

Enunciado: Calcular el promedio de los números del 1 al N que sean impares.

Código:

```
# 95 - Promedio de impares
n = int(input("Introduce N: "))
suma = 0
contador = 0
for i in range(1, n+1, 2):
    suma += i
    contador += 1
if contador > 0:
    print("Promedio de impares:", suma/contador)
```

Explicación: Recorre solo los impares y calcula la media.

96.

Enunciado: Verificar si un número es divisible entre 2 y 3 al mismo tiempo.
Código:

```
# 96 - Divisible entre 2 y 3
n = int(input("Introduce un número: "))
if n % 2 == 0 and n % 3 == 0:
    print(f"{n} es divisible entre 2 y 3")
else:
    print(f"{n} no es divisible entre 2 y 3")
```

Explicación: Combina condiciones con `and` para verificar divisibilidad múltiple.

97.

Enunciado: Mostrar los primeros N números positivos.
Código:

```
# 97 - Primeros N números
n = int(input("Introduce N: "))
for i in range(1, n+1):
    print(i, end=" ")
```

Explicación: Bucle simple desde 1 hasta N.

98.

Enunciado: Calcular la suma de los primeros N números impares.
Código:

```
# 98 - Suma de N impares
n = int(input("Introduce N: "))
suma = 0
for i in range(n):
    suma += 2*i + 1
print("Suma de los primeros N impares:", suma)
```

Explicación: Usa la fórmula `2*i+1` para obtener los impares consecutivos y acumularlos.

99.

Enunciado: Mostrar los números del 1 al N que sean múltiplos de 2 o de 5.
Código:

```
# 99 - Múltiplos de 2 o 5
n = int(input("Introduce N: "))
for i in range(1, n+1):
    if i % 2 == 0 or i % 5 == 0:
        print(i, end=" ")
```

Explicación: Filtra los números que cumplan al menos una de las dos condiciones.

100.

Enunciado: Calcular la suma de los primeros N números pares.

Código:

```
# 100 - Suma de N primeros pares
n = int(input("Introduce N: "))
suma = 0
for i in range(n):
    suma += 2*i
print("Suma de los primeros N números pares:", suma)
```

Explicación: Genera los N primeros pares con `2*i` y acumula la suma.

Ejercicios python con cadenas de texto

Programa 1: Mostrar una cadena

Enunciado:
Pide al usuario una cadena y la muestra en pantalla.

```
texto = input("Introduce un texto: ")
print("Has escrito:", texto)
```

Explicación:
El programa solicita al usuario que escriba algo y luego muestra lo que introdujo usando la función `print()`.

Programa 2: Calcular la longitud de una cadena

Enunciado:
Pide una palabra y muestra cuántos caracteres tiene.

```
palabra = input("Introduce una palabra: ")
print("La palabra tiene", len(palabra), "caracteres.")
```

Explicación:
La función `len()` devuelve el número total de caracteres de la cadena.

Programa 3: Convertir a mayúsculas

Enunciado:
Convierte una cadena a mayúsculas.

```
texto = input("Escribe algo: ")
print("En mayúsculas:", texto.upper())
```

Explicación:
El método `.upper()` transforma todas las letras del texto a mayúsculas.

Programa 4: Convertir a minúsculas

Enunciado:
Convierte un texto a minúsculas.

```
texto = input("Escribe algo: ")
print("En minúsculas:", texto.lower())
```

Explicación:
El método `.lower()` convierte todas las letras a minúsculas.

Programa 5: Capitalizar un texto

Enunciado:
Convierte el primer carácter a mayúscula y el resto a minúsculas.

```
texto = input("Introduce una frase: ")
print("Capitalizado:", texto.capitalize())
```

Explicación:
El método `.capitalize()` solo pone en mayúscula la primera letra del texto.

Programa 6: Contar una letra específica

Enunciado:
Cuenta cuántas veces aparece una letra elegida en un texto.

```
texto = input("Escribe un texto: ")
letra = input("Qué letra quieres contar?: ")
print("La letra", letra, "aparece", texto.count(letra), "veces.")
```

Explicación:
El método `.count()` devuelve cuántas veces aparece una subcadena dentro del texto.

Programa 7: Reemplazar palabras

Enunciado:
Reemplaza una palabra por otra dentro de un texto.

```
texto = input("Escribe una frase: ")
buscar = input("Palabra a reemplazar: ")
nueva = input("Nueva palabra: ")
print("Resultado:", texto.replace(buscar, nueva))
```

Explicación:
El método `.replace()` sustituye todas las ocurrencias de una palabra por otra.

Programa 8: Verificar si contiene una palabra

Enunciado:
Comprueba si un texto contiene una palabra determinada.

```
texto = input("Escribe un texto: ")
palabra = input("Palabra a buscar: ")
if palabra in texto:
    print("El texto contiene la palabra.")
else:
    print("La palabra no está en el texto.")
```

Explicación:
El operador `in` permite comprobar si una subcadena está contenida en otra.

Programa 9: Invertir una cadena

Enunciado:
Muestra el texto al revés.

```
texto = input("Escribe un texto: ")
print("Texto invertido:", texto[::-1])
```

Explicación:
La notación de corte `[::-1]` recorre la cadena desde el final hasta el principio.

Programa 10: Contar vocales

Enunciado:
Cuenta cuántas vocales tiene un texto.

```
texto = input("Introduce un texto: ").lower()
contador = 0
for letra in texto:
    if letra in "aeiou":
        contador += 1
print("Número de vocales:", contador)
```

Explicación:
Convierte el texto a minúsculas y recorre cada letra; si es una vocal, suma 1 al contador.

Programa 11: Unir dos cadenas

Enunciado:
Pide dos palabras y muestra el resultado de unirlas.

```
palabra1 = input("Primera palabra: ")
palabra2 = input("Segunda palabra: ")
resultado = palabra1 + palabra2
print("Resultado:", resultado)
```

Explicación:
El operador `+` concatena (une) dos cadenas sin espacios adicionales.

Programa 12: Unir dos cadenas con espacio

Enunciado:
Concatena dos palabras separándolas con un espacio.

```
palabra1 = input("Primera palabra: ")
palabra2 = input("Segunda palabra: ")
resultado = palabra1 + " " + palabra2
print("Resultado:", resultado)
```

Explicación:
Se agrega un espacio manualmente al unir las dos palabras.

Programa 13: Verificar si una cadena empieza con una letra

Enunciado:
Comprueba si un texto empieza por una letra dada.

```
texto = input("Introduce un texto: ")
letra = input("Introduce una letra: ")
if texto.startswith(letra):
    print("El texto empieza con esa letra.")
else:
    print("El texto no empieza con esa letra.")
```

Explicación:
El método `.startswith()` devuelve `True` si el texto comienza con la subcadena indicada.

Programa 14: Verificar si una cadena termina con una letra

Enunciado:
Comprueba si una cadena termina en una letra o palabra determinada.

```
texto = input("Introduce un texto: ")
termina = input("Introduce una letra o palabra: ")
if texto.endswith(termina):
    print("El texto termina con eso.")
else:
    print("El texto no termina con eso.")
```

Explicación:
El método `.endswith()` verifica si el texto finaliza con la subcadena indicada.

Programa 15: Contar palabras en una frase

Enunciado:
Cuenta cuántas palabras tiene una frase.

```
frase = input("Escribe una frase: ")
palabras = frase.split()
print("Número de palabras:", len(palabras))
```

Explicación:
El método `.split()` divide la frase en una lista de palabras separadas por espacios, y `len()` cuenta cuántas hay.

Programa 16: Separar las letras de una palabra

Enunciado:
Muestra cada letra de una palabra en una línea diferente.

```
palabra = input("Introduce una palabra: ")
for letra in palabra:
    print(letra)
```

Explicación:
El bucle `for` recorre cada carácter de la cadena y lo imprime uno por línea.

Programa 17: Repetir una palabra varias veces

Enunciado:
Pide una palabra y un número, y repite la palabra esa cantidad de veces.

```
palabra = input("Introduce una palabra: ")
n = int(input("¿Cuántas veces quieres repetirla?: "))
print(palabra * n)
```

Explicación:
El operador `*` permite repetir una cadena tantas veces como se indique.

Programa 18: Quitar espacios al principio y al final

Enunciado:
Elimina los espacios sobrantes de una cadena.

```
texto = input("Escribe un texto con espacios: ")
print("Texto sin espacios:", texto.strip())
```

Explicación:
El método `.strip()` borra los espacios al inicio y final del texto.

Programa 19: Quitar solo los espacios del principio

Enunciado:
Elimina los espacios que hay al inicio de una cadena.

```
texto = input("Escribe un texto con espacios al principio: ")
print("Resultado:", texto.lstrip())
```

Explicación:
El método `.lstrip()` borra los espacios solo del lado izquierdo de la cadena.

Programa 20: Quitar solo los espacios del final

Enunciado:
Elimina los espacios que hay al final de una cadena.

```
texto = input("Escribe un texto con espacios al final: ")
print("Resultado:", texto.rstrip())
```

Explicación:
El método `.rstrip()` borra los espacios solo del lado derecho de la cadena.

Programa 21: Convertir una frase en lista de palabras

Enunciado:
Convierte una frase en una lista de palabras separadas.

```
frase = input("Escribe una frase: ")
palabras = frase.split()
print("Lista de palabras:", palabras)
```

Explicación:
El método `.split()` divide la frase en una lista, tomando los espacios como separadores por defecto.

Programa 22: Unir lista de palabras en una sola cadena

Enunciado:
Une una lista de palabras en una sola cadena separadas por espacios.

```
palabras = ["Hola", "mundo", "Python"]
frase = " ".join(palabras)
print("Frase unida:", frase)
```

Explicación:
El método `.join()` une los elementos de una lista en una sola cadena, usando el separador indicado antes del `.join()`.

Programa 23: Reemplazar todas las vocales por asteriscos

Enunciado:
Cambia todas las vocales de un texto por el símbolo *

```
texto = input("Escribe un texto: ")
nuevo = ""
for letra in texto:
    if letra.lower() in "aeiou":
        nuevo += "*"
    else:
        nuevo += letra
print("Texto modificado:", nuevo)
```

Explicación:
El programa recorre cada letra y reemplaza las vocales por `*`, acumulando el resultado en una nueva cadena.

Programa 24: Comprobar si una palabra es mayúscula

Enunciado:
Verifica si toda una palabra está en mayúsculas.

```
palabra = input("Introduce una palabra: ")
if palabra.isupper():
    print("La palabra está en mayúsculas.")
else:
    print("No está en mayúsculas.")
```

Explicación:
El método `.isupper()` devuelve `True` si todos los caracteres son letras mayúsculas.

Programa 25: Comprobar si una palabra es minúscula

Enunciado:
Verifica si toda una palabra está en minúsculas.

```
palabra = input("Introduce una palabra: ")
if palabra.islower():
    print("La palabra está en minúsculas.")
else:
    print("No está en minúsculas.")
```

Explicación:
El método `.islower()` devuelve `True` si todos los caracteres son letras minúsculas.

Programa 26: Verificar si una cadena contiene solo letras

Enunciado:
Determina si el texto contiene solo caracteres alfabéticos.

```
texto = input("Introduce un texto: ")
if texto.isalpha():
    print("El texto contiene solo letras.")
else:
    print("Contiene otros caracteres además de letras.")
```

Explicación:
El método `.isalpha()` devuelve `True` si todos los caracteres son letras sin espacios ni números.

Programa 27: Verificar si una cadena contiene solo números

Enunciado:
Comprueba si el texto está formado únicamente por números.

```
texto = input("Introduce un texto: ")
if texto.isdigit():
    print("El texto contiene solo números.")
else:
    print("El texto tiene otros caracteres además de números.")
```

Explicación:
El método `.isdigit()` devuelve `True` si todos los caracteres son dígitos numéricos.

Programa 28: Contar espacios en una frase

Enunciado:
Cuenta cuántos espacios hay en una frase.

```
frase = input("Introduce una frase: ")
espacios = frase.count(" ")
print("Número de espacios:", espacios)
```

Explicación:
El método `.count(" ")` cuenta cuántos espacios hay dentro del texto.

Programa 29: Mostrar solo las letras mayúsculas de una frase

Enunciado:
Muestra únicamente las letras que estén en mayúsculas dentro de una frase.

```
frase = input("Escribe una frase: ")
mayusculas = ""
for letra in frase:
    if letra.isupper():
        mayusculas += letra
print("Letras mayúsculas:", mayusculas)
```

Explicación:
El programa recorre cada carácter y añade a la nueva cadena solo las letras mayúsculas.

Programa 30: Comprobar si una palabra es un palíndromo

Enunciado:
Determina si una palabra se lee igual al derecho y al revés.

```
palabra = input("Introduce una palabra: ").lower()
if palabra == palabra[::-1]:
    print("Es un palíndromo.")
else:
    print("No es un palíndromo.")
```

Explicación:
Se compara la palabra original con su versión invertida (`[::-1]`). Si son iguales, es palíndroma.

Programa 31: Mostrar las posiciones de una letra en el texto

Enunciado:
Indica en qué posiciones aparece una letra dentro de una cadena.

```
texto = input("Introduce un texto: ")
letra = input("Letra a buscar: ")
for i in range(len(texto)):
    if texto[i] == letra:
        print("Letra encontrada en posición:", i)
```

Explicación:
Se recorre el texto por índice con `range(len(texto))` y se muestra la posición donde aparece la letra buscada.

Programa 32: Repetir cada letra de una palabra dos veces

Enunciado:

Crea una nueva palabra repitiendo cada letra dos veces.

```
palabra = input("Introduce una palabra: ")
nueva = ""
for letra in palabra:
    nueva += letra * 2
print("Resultado:", nueva)
```

Explicación:

Cada letra se multiplica por 2 para duplicarla en la nueva cadena.

Programa 33: Contar consonantes en un texto

Enunciado:

Cuenta cuántas consonantes contiene una cadena.

```
texto = input("Escribe un texto: ").lower()
contador = 0
for letra in texto:
    if letra in "bcdfghjklmnpqrstvwxyz":
        contador += 1
print("Número de consonantes:", contador)
```

Explicación:

Convierte el texto a minúsculas y suma 1 cada vez que la letra es una consonante.

Programa 34: Reemplazar espacios por guiones

Enunciado:

Transforma todos los espacios de un texto en guiones medios.

```
texto = input("Escribe una frase: ")
print("Texto modificado:", texto.replace(" ", "-"))
```

Explicación:

El método `.replace(" ", "-")` sustituye los espacios por guiones.

Programa 35: Mostrar los tres primeros caracteres de un texto

Enunciado:

Muestra únicamente los tres primeros caracteres introducidos.

```
texto = input("Introduce un texto: ")
print("Primeros tres caracteres:", texto[:3])
```

Explicación:

El corte `[:3]` toma los caracteres desde la posición 0 hasta la 2 (sin incluir la 3).

Programa 36: Mostrar los tres últimos caracteres de un texto

Enunciado:
Muestra solo los tres últimos caracteres de la cadena.

```
texto = input("Introduce un texto: ")
print("Últimos tres caracteres:", texto[-3:])
```

Explicación:
El corte `[-3:]` obtiene los últimos tres caracteres del texto.

Programa 37: Eliminar todas las vocales

Enunciado:
Crea una nueva cadena sin vocales.

```
texto = input("Escribe un texto: ")
sin_vocales = ""
for letra in texto:
    if letra.lower() not in "aeiou":
        sin_vocales += letra
print("Texto sin vocales:", sin_vocales)
```

Explicación:
El programa recorre el texto y agrega solo los caracteres que no sean vocales.

Programa 38: Contar cuántas veces aparece cada letra

Enunciado:
Muestra la cantidad de veces que aparece cada letra del texto.

```
texto = input("Introduce un texto: ").lower()
for letra in set(texto):
    if letra.isalpha():
        print(letra, ":", texto.count(letra))
```

Explicación:
El uso de `set(texto)` evita repetir letras. Luego se cuentan con `.count()` solo las que son alfabéticas.

Programa 39: Sustituir las letras por su posición numérica

Enunciado:
Convierte cada letra en su número de posición alfabética (a=1, b=2...).

```
texto = input("Escribe una palabra: ").lower()
for letra in texto:
    if letra.isalpha():
        print(letra, "=", ord(letra) - 96)
```

Explicación:
La función `ord()` devuelve el código ASCII; restando 96 se obtiene la posición en el alfabeto (para minúsculas).

Programa 40: Intercalar guiones entre letras

Enunciado:
Inserta un guion entre cada letra de una palabra.

```
palabra = input("Introduce una palabra: ")
resultado = "-".join(palabra)
print("Resultado:", resultado)
```

Explicación:
El método `.join()` coloca el guion `-` entre cada carácter de la cadena.

Programa 41: Repetir frase separada por saltos de línea

Enunciado:
Pide una frase y un número, y muestra la frase repetida en distintas líneas.

```
frase = input("Introduce una frase: ")
n = int(input("¿Cuántas veces quieres repetirla?: "))
for i in range(n):
    print(frase)
```

Explicación:
El bucle `for` imprime la misma frase `n` veces, cada una en una línea nueva.

Programa 42: Mostrar el carácter central de una palabra

Enunciado:
Muestra el carácter que está justo en el centro de una palabra.

```
palabra = input("Introduce una palabra: ")
posicion = len(palabra) // 2
print("Carácter central:", palabra[posicion])
```

Explicación:
Se calcula la posición media con `len(palabra)//2`, usando división entera.

Programa 43: Convertir la primera letra de cada palabra en mayúscula

Enunciado:
Convierte la primera letra de cada palabra de una frase en mayúscula.

```
frase = input("Introduce una frase: ")
print("Resultado:", frase.title())
```

Explicación:
El método `.title()` convierte la primera letra de cada palabra a mayúscula.

Programa 44: Eliminar todos los signos de puntuación

Enunciado:
Crea una versión del texto sin signos de puntuación.

```
texto = input("Escribe un texto: ")
puntuacion = ".,;:!?¿¡()\"'-"
nuevo = ""
for letra in texto:
    if letra not in puntuacion:
        nuevo += letra
print("Texto sin puntuación:", nuevo)
```

Explicación:
Se recorren los caracteres del texto y se copian solo los que no están en la lista de signos.

Programa 45: Comparar dos cadenas ignorando mayúsculas

Enunciado:
Comprueba si dos textos son iguales sin importar las mayúsculas o minúsculas.

```
texto1 = input("Primer texto: ").lower()
texto2 = input("Segundo texto: ").lower()
if texto1 == texto2:
    print("Los textos son iguales.")
else:
    print("Los textos son diferentes.")
```

Explicación:
Ambos textos se convierten a minúsculas para realizar una comparación insensible a mayúsculas.

Programa 46: Extraer números de un texto

Enunciado:
Obtiene solo los números presentes en una cadena.

```
texto = input("Escribe un texto con números: ")
numeros = ""
for letra in texto:
    if letra.isdigit():
        numeros += letra
print("Números encontrados:", numeros)
```

Explicación:
El método `.isdigit()` permite identificar los caracteres numéricos dentro de un texto.

Programa 47: Eliminar números de un texto

Enunciado:
Elimina todos los números y deja solo las letras y símbolos.

```
texto = input("Escribe un texto con números: ")
solo_letras = ""
for letra in texto:
    if not letra.isdigit():
        solo_letras += letra
print("Texto sin números:", solo_letras)
```

Explicación:
Se usa `not letra.isdigit()` para ignorar los números durante la copia del texto.

Programa 48: Mostrar las palabras ordenadas alfabéticamente

Enunciado:
Ordena las palabras de una frase por orden alfabético.

```
frase = input("Escribe una frase: ")
palabras = frase.split()
palabras.sort()
print("Palabras ordenadas:", palabras)
```

Explicación:
El método `.split()` separa las palabras y `.sort()` las organiza en orden alfabético.

Programa 49: Contar letras y números en un texto

Enunciado:
Cuenta cuántas letras y cuántos números hay en una cadena.

```
texto = input("Introduce un texto: ")
letras = 0
numeros = 0
for c in texto:
    if c.isalpha():
        letras += 1
    elif c.isdigit():
        numeros += 1
print("Letras:", letras)
print("Números:", numeros)
```

Explicación:
Se recorren los caracteres y se usa `.isalpha()` y `.isdigit()` para contarlos según su tipo.

Programa 50: Invertir el orden de las palabras

Enunciado:
Invierte el orden de las palabras de una frase.

```
frase = input("Escribe una frase: ")
palabras = frase.split()
palabras.reverse()
print("Frase invertida:", " ".join(palabras))
```

Explicación:
Primero se divide la frase en palabras con `.split()`, luego se invierte con `.reverse()` y se vuelve a unir con `.join()`.

Programa 51: Contar cuántas veces aparece una palabra

Enunciado:
Cuenta cuántas veces se repite una palabra específica dentro de un texto.

```
texto = input("Escribe un texto: ").lower()
palabra = input("Palabra a buscar: ").lower()
print("La palabra aparece", texto.split().count(palabra), "veces.")
```

Explicación:
Se convierte todo a minúsculas para evitar errores y se usa `.split().count()` para contar las repeticiones exactas.

Programa 52: Verificar si una frase es palíndroma ignorando espacios

Enunciado:
Comprueba si una frase se lee igual al derecho y al revés, sin importar espacios ni mayúsculas.

```
frase = input("Introduce una frase: ").replace(" ", "").lower()
if frase == frase[::-1]:
    print("Es palíndroma.")
else:
    print("No es palíndroma.")
```

Explicación:
Se eliminan los espacios con `.replace()` y se compara la frase con su versión invertida.

Programa 53: Mostrar caracteres únicos de una palabra

Enunciado:
Muestra las letras que aparecen una sola vez en la palabra.

```
palabra = input("Introduce una palabra: ").lower()
for letra in palabra:
    if palabra.count(letra) == 1:
        print(letra, end=" ")
```

Explicación:
El método `.count()` verifica cuántas veces aparece cada letra; si es 1, se imprime.

Programa 54: Mostrar solo los dígitos pares de un texto

Enunciado:
Extrae y muestra solo los números pares dentro de un texto.

```
texto = input("Escribe algo con números: ")
for c in texto:
    if c.isdigit() and int(c) % 2 == 0:
        print(c, end=" ")
```

Explicación:
Se comprueba con `.isdigit()` si el carácter es número y luego si es par (`% 2 == 0`).

Programa 55: Sustituir las vocales por números

Enunciado:
Cambia las vocales por números según su orden: a=1, e=2, i=3, o=4, u=5.

```
texto = input("Introduce un texto: ").lower()
conversion = {"a":"1","e":"2","i":"3","o":"4","u":"5"}
nuevo = ""
for letra in texto:
    nuevo += conversion.get(letra, letra)
print("Resultado:", nuevo)
```

Explicación:
Se usa un diccionario `conversion` y el método `.get()` para reemplazar las vocales por números.

Programa 56: Comprobar si una cadena contiene solo espacios

Enunciado:
Indica si el texto ingresado está compuesto únicamente por espacios.

```
texto = input("Introduce un texto: ")
if texto.isspace():
    print("Solo contiene espacios.")
else:
    print("Contiene otros caracteres.")
```

Explicación:
El método `.isspace()` devuelve `True` si todos los caracteres son espacios.

Programa 57: Reemplazar cada espacio por una coma

Enunciado:
Transforma todos los espacios de una frase en comas.

```
texto = input("Introduce una frase: ")
print("Texto modificado:", texto.replace(" ", ","))
```

Explicación:
El método `.replace(" ", ",")` cambia todos los espacios por comas.

Programa 58: Mostrar letras alternas de una palabra

Enunciado:
Muestra una palabra formada solo por las letras en posiciones pares.

```
palabra = input("Introduce una palabra: ")
print("Letras alternas:", palabra[::2])
```

Explicación:
El corte `[::2]` toma un carácter y salta el siguiente, mostrando las posiciones 0, 2, 4, etc.

Programa 59: Convertir números de texto a enteros y sumarlos

Enunciado:
Lee dos números escritos como texto, los convierte a enteros y los suma.

```
n1 = input("Introduce el primer número: ")
n2 = input("Introduce el segundo número: ")
suma = int(n1) + int(n2)
print("La suma es:", suma)
```

Explicación:
Los valores de entrada son cadenas, y se convierten con `int()` para poder sumarlos.

Programa 60: Mostrar solo los caracteres alfabéticos

Enunciado:
Filtra una cadena dejando solo las letras.

```
texto = input("Introduce un texto: ")
solo_letras = ""
for c in texto:
    if c.isalpha():
        solo_letras += c
print("Solo letras:", solo_letras)
```

Explicación:
Con `.isalpha()` se seleccionan solo los caracteres que son letras.

Programa 61: Contar letras mayúsculas y minúsculas

Enunciado:
Cuenta cuántas letras están en mayúscula y cuántas en minúscula dentro de un texto.

```
texto = input("Introduce un texto: ")
mayus = 0
minus = 0
for c in texto:
    if c.isupper():
        mayus += 1
    elif c.islower():
        minus += 1
print("Mayúsculas:", mayus)
print("Minúsculas:", minus)
```

Explicación:
Los métodos `.isupper()` y `.islower()` permiten distinguir las letras según su tipo.

Programa 62: Insertar un carácter cada cierta cantidad de letras

Enunciado:
Pide un texto y un número, e inserta un guion cada n caracteres.

```
texto = input("Introduce un texto: ")
n = int(input("Cada cuántas letras quieres insertar un guion?: "))
resultado = ""
for i in range(len(texto)):
    resultado += texto[i]
    if (i + 1) % n == 0:
        resultado += "-"
print("Resultado:", resultado)
```

Explicación:
Usa el operador módulo `%` para insertar un guion tras cada grupo de `n` letras.

Programa 63: Eliminar una letra específica de una cadena

Enunciado:
Pide una letra y elimina todas sus apariciones en un texto.

```
texto = input("Introduce un texto: ")
letra = input("Qué letra quieres eliminar?: ")
print("Texto sin esa letra:", texto.replace(letra, ""))
```

Explicación:
El método `.replace()` sustituye las apariciones de la letra indicada por una cadena vacía.

Programa 64: Convertir una cadena en una lista de caracteres

Enunciado:
Convierte cada letra de un texto en un elemento de lista.

```
texto = input("Introduce un texto: ")
lista = list(texto)
print("Lista de caracteres:", lista)
```

Explicación:
La función `list()` convierte cualquier cadena en una lista donde cada carácter es un elemento independiente.

Programa 65: Unir lista de caracteres en una cadena

Enunciado:
Toma una lista de letras y las une formando una palabra.

```
lista = ['H', 'o', 'l', 'a']
texto = "".join(lista)
print("Palabra formada:", texto)
```

Explicación:
El método `.join()` une todos los elementos de la lista sin espacio entre ellos.

Programa 66: Mostrar la letra con mayor valor alfabético

Enunciado:
Indica cuál es la letra más “grande” en orden alfabético.

```
texto = input("Introduce una palabra: ").lower()
print("Letra más alta:", max(texto))
```

Explicación:
La función `max()` devuelve el carácter con mayor valor según el orden alfabético.

Programa 67: Mostrar la letra con menor valor alfabético

Enunciado:
Muestra la primera letra en orden alfabético de un texto.

```
texto = input("Introduce una palabra: ").lower()
print("Letra más baja:", min(texto))
```

Explicación:
La función `min()` devuelve el carácter más pequeño en el orden alfabético.

Programa 68: Mostrar la palabra más larga de una frase

Enunciado:
Muestra cuál es la palabra más larga de una frase dada.

```
frase = input("Introduce una frase: ")
palabras = frase.split()
larga = max(palabras, key=len)
print("Palabra más larga:", larga)
```

Explicación:
La función `max()` con el parámetro `key=len` devuelve el elemento más largo según su longitud.

Programa 69: Mostrar la palabra más corta de una frase

Enunciado:
Muestra la palabra con menos letras de una frase.

```
frase = input("Introduce una frase: ")
palabras = frase.split()
corta = min(palabras, key=len)
print("Palabra más corta:", corta)
```

Explicación:
La función `min()` con `key=len` devuelve el elemento más corto de la lista de palabras.

Programa 70: Eliminar las letras repetidas

Enunciado:
Crea una cadena sin caracteres duplicados, conservando el orden.

```
texto = input("Introduce un texto: ")
nuevo = ""
for letra in texto:
    if letra not in nuevo:
        nuevo += letra
print("Texto sin duplicados:", nuevo)
```

Explicación:
El programa va construyendo una nueva cadena agregando solo las letras que aún no han aparecido.

Programa 71: Contar cuántas veces aparece cada palabra

Enunciado:
Cuenta cuántas veces aparece cada palabra dentro de una frase.

```
frase = input("Introduce una frase: ").lower()

palabras = frase.split()

for p in set(palabras):

    print(p, ":", palabras.count(p))
```

Explicación:
`set(palabras)` evita repeticiones, y `.count(p)` muestra cuántas veces aparece cada palabra.

Programa 72: Mostrar solo los caracteres únicos del texto

Enunciado:
Muestra los caracteres que no se repiten en todo el texto.

```
texto = input("Introduce un texto: ").lower()
for c in texto:
    if texto.count(c) == 1:
        print(c, end=" ")
```

Explicación:
Se recorre el texto y se imprime el carácter si aparece exactamente una vez.

Programa 73: Dividir un texto en frases

Enunciado:
Divide un texto en frases separadas por puntos.

```
texto = input("Introduce un texto con varias frases: ")
frases = texto.split(".")
for f in frases:
    print(f.strip())
```

Explicación:
Se usa `.split(".")` para separar el texto y `.strip()` para eliminar espacios sobrantes.

Programa 74: Sustituir vocales por mayúsculas

Enunciado:
Convierte las vocales del texto en su versión mayúscula.

```
texto = input("Introduce un texto: ")
nuevo = ""
for letra in texto:
    if letra.lower() in "aeiou":
        nuevo += letra.upper()
    else:
        nuevo += letra
print("Resultado:", nuevo)
```

Explicación:
El programa convierte las vocales a mayúsculas manteniendo el resto igual.

Programa 75: Insertar un espacio entre cada letra

Enunciado:
Agrega un espacio entre todos los caracteres de un texto.

```
texto = input("Introduce un texto: ")
print("Resultado:", " ".join(texto))
```

Explicación:
El método `.join()` une las letras usando un espacio como separador.

Programa 76: Verificar si todas las palabras empiezan con mayúscula

Enunciado:
Comprueba si cada palabra de la frase comienza con mayúscula.

```
frase = input("Introduce una frase: ")
palabras = frase.split()
correcto = all(p[0].isupper() for p in palabras)
print("¿Todas comienzan con mayúscula?:", correcto)
```

Explicación:
Se usa `all()` para verificar si todas las primeras letras (`p[0]`) son mayúsculas.

Programa 77: Contar cuántas letras hay en total sin espacios

Enunciado:
Cuenta todas las letras del texto ignorando los espacios.

```
texto = input("Introduce un texto: ")
sin_espacios = texto.replace(" ", "")
print("Total de letras:", len(sin_espacios))
```

Explicación:
Se eliminan los espacios con `.replace(" ", "")` y se cuenta con `len()`.

Programa 78: Mostrar el texto en forma de triángulo

Enunciado:
Muestra el texto de manera creciente, añadiendo una letra por línea.

```
texto = input("Introduce una palabra: ")
for i in range(1, len(texto)+1):
    print(texto[:i])
```

Explicación:
En cada iteración se imprimen los primeros `i` caracteres, creando un efecto triangular.

Programa 79: Mostrar el texto en forma de triángulo invertido

Enunciado:
Imprime la palabra reduciendo una letra en cada línea.

```
texto = input("Introduce una palabra: ")
for i in range(len(texto), 0, -1):
    print(texto[:i])
```

Explicación:
El bucle `range()` empieza desde el final y reduce la longitud del texto en cada línea.

Programa 80: Detectar palabras que empiezan con una letra específica

Enunciado:
Muestra las palabras de una frase que comienzan con una letra dada.

```
frase = input("Introduce una frase: ")
letra = input("Letra inicial a buscar: ").lower()
for palabra in frase.split():
    if palabra.lower().startswith(letra):
        print(palabra)
```

Explicación:
El método `.startswith()` identifica las palabras que comienzan con la letra elegida.

Programa 81: Contar cuántas palabras hay en una frase

Enunciado:
Cuenta cuántas palabras contiene una frase escrita por el usuario.

```
frase = input("Introduce una frase: ")
palabras = frase.split()
print("Número de palabras:", len(palabras))
```

Explicación:
El método `.split()` divide el texto en una lista de palabras, y `len()` cuenta cuántas hay.

Programa 82: Verificar si una cadena está vacía

Enunciado:
Indica si el texto introducido está vacío o no.

```
texto = input("Introduce un texto: ")
if texto.strip() == "":
    print("La cadena está vacía.")
else:
    print("La cadena contiene texto.")
```

Explicación:
`.strip()` elimina espacios al principio y final antes de comprobar si queda texto.

Programa 83: Repetir una cadena varias veces

Enunciado:
Pide una palabra y un número y repite la palabra ese número de veces.

```
texto = input("Introduce una palabra: ")
n = int(input("Cuántas veces quieres repetirla?: "))
print(texto * n)
```

Explicación:
En Python, el operador `*` permite repetir una cadena varias veces seguidas.

Programa 84: Mostrar la posición de todas las apariciones de una letra

Enunciado:
Indica en qué posiciones aparece una letra dentro de un texto.

```
texto = input("Introduce un texto: ")
letra = input("Qué letra quieres buscar?: ")
for i in range(len(texto)):
    if texto[i] == letra:
        print("Posición:", i)
```

Explicación:
Se recorre la cadena carácter por carácter mostrando las posiciones donde coincide la letra.

Programa 85: Mostrar las palabras en orden alfabético

Enunciado:
Muestra las palabras de una frase ordenadas alfabéticamente.

```
frase = input("Introduce una frase: ")
palabras = frase.split()
palabras.sort()
print("Palabras ordenadas:", palabras)
```

Explicación:
El método `.sort()` ordena la lista de palabras en orden alfabético.

Programa 86: Invertir el orden de las palabras en una frase

Enunciado:
Muestra las palabras de una frase en orden inverso.

```
frase = input("Introduce una frase: ")
palabras = frase.split()
palabras.reverse()
print("Frase invertida:", " ".join(palabras))
```

Explicación:
`.reverse()` invierte el orden de la lista y `.join()` la vuelve a unir en una cadena.

Programa 87: Reemplazar todas las vocales por un símbolo

Enunciado:
Cambia todas las vocales del texto por el carácter `*`.

```
texto = input("Introduce un texto: ")
nuevo = ""
for letra in texto:
    if letra.lower() in "aeiou":
        nuevo += "*"
    else:
        nuevo += letra
print("Resultado:", nuevo)
```

Explicación:
Se recorre cada letra y se sustituye por `*` si es una vocal.

Programa 88: Mostrar solo los dígitos de una cadena

Enunciado:
Extrae únicamente los números que aparecen dentro del texto.

```
texto = input("Introduce un texto con números: ")
numeros = ""
for c in texto:
    if c.isdigit():
        numeros += c
print("Dígitos encontrados:", numeros)
```

Explicación:
El método `.isdigit()` permite identificar los caracteres numéricos.

Programa 89: Eliminar los signos de puntuación

Enunciado:
Muestra el texto sin puntos, comas ni signos de exclamación o interrogación.

```
texto = input("Introduce un texto: ")
puntuacion = ".,;:!?¡¿"
for signo in puntuacion:
    texto = texto.replace(signo, "")
print("Texto limpio:", texto)
```

Explicación:
El bucle recorre cada signo de puntuación y lo reemplaza por una cadena vacía.

Programa 90: Verificar si una palabra es palíndromo ignorando espacios

Enunciado:
Comprueba si una frase es palíndromo sin tener en cuenta los espacios.

```
frase = input("Introduce una frase: ").replace(" ", "").lower()
if frase == frase[::-1]:
    print("Es palíndromo.")
else:
    print("No es palíndromo.")
```

Explicación:
Elimina los espacios, pasa todo a minúsculas y compara la frase con su reverso.

Programa 91: Convertir toda la frase a mayúsculas

Enunciado:
Convierte todo el texto introducido por el usuario en letras mayúsculas.

```
texto = input("Introduce un texto: ")

print("Mayúsculas:", texto.upper())
```

Explicación:
El método `.upper()` transforma todas las letras de la cadena a mayúsculas.

Programa 92: Convertir toda la frase a minúsculas

Enunciado:
Convierte todo el texto a letras minúsculas.

```
texto = input("Introduce un texto: ")

print("Minúsculas:", texto.lower())
```

Explicación:
El método `.lower()` transforma todas las letras de la cadena a minúsculas.

Programa 93: Capitalizar la primera letra de la frase

Enunciado:
Convierte la primera letra de la frase en mayúscula y el resto en minúsculas.

```
texto = input("Introduce un texto: ")
print("Capitalizado:", texto.capitalize())
```

Explicación:
El método `.capitalize()` pone en mayúscula la primera letra de la cadena y el resto en minúsculas.

Programa 94: Contar cuántas letras tiene la frase

Enunciado:
Cuenta la cantidad de caracteres de la frase incluyendo espacios.

```
texto = input("Introduce un texto: ")
print("Número de caracteres:", len(texto))
```

Explicación:
La función `len()` devuelve el número total de caracteres en la cadena.

Programa 95: Contar cuántos espacios hay en la frase

Enunciado:
Cuenta la cantidad de espacios en un texto.

```
texto = input("Introduce un texto: ")
print("Número de espacios:", texto.count(" "))
```

Explicación:
El método `.count(" ")` cuenta cuántas veces aparece un espacio en la cadena.

Programa 96: Contar cuántas vocales hay en la frase

Enunciado:
Cuenta cuántas vocales tiene un texto.

```
texto = input("Introduce un texto: ").lower()
contador = 0
for letra in texto:
    if letra in "aeiou":
        contador += 1
print("Número de vocales:", contador)
```

Explicación:
Se recorre la cadena y se incrementa el contador cada vez que se encuentra una vocal.

Programa 97: Contar cuántas consonantes hay en la frase

Enunciado:
Cuenta las consonantes dentro de un texto.

```
texto = input("Introduce un texto: ").lower()
contador = 0
for letra in texto:
    if letra.isalpha() and letra not in "aeiou":
        contador += 1
print("Número de consonantes:", contador)
```

Explicación:
Se comprueba que sea letra y no sea vocal para contar las consonantes.

Programa 98: Sustituir todas las letras “a” por “@”

Enunciado:
Cambia todas las letras `a` del texto por el símbolo `@`.

```
texto = input("Introduce un texto: ")
print("Texto modificado:", texto.replace("a", "@").replace("A", "@"))
```

Explicación:
Se usa `.replace()` para cambiar todas las `a` y `A` por `@`.

Programa 99: Verificar si la frase contiene una palabra

Enunciado:
Comprueba si una palabra está dentro de un texto.

```
texto = input("Introduce un texto: ").lower()
palabra = input("Palabra a buscar: ").lower()
if palabra in texto:
    print("La palabra está en el texto.")
else:
    print("La palabra no se encuentra.")
```

Explicación:
El operador `in` verifica si una subcadena aparece dentro de otra cadena.

Programa 100: Mostrar el texto invertido

Enunciado:
Muestra el texto al revés, es decir, desde el último carácter hasta el primero.

```
texto = input("Introduce un texto: ")
print("Texto invertido:", texto[::-1])
```

Explicación:
El corte `[::-1]` invierte la cadena mostrando los caracteres en orden inverso.

Ejercicios python estructuras de datos, listas

Ejercicio 1

Enunciado: Crea una lista con los números del 1 al 5 y muéstrala por pantalla.
Explicación: Se trata de crear una lista básica en Python y mostrar su contenido usando `print()`.

```
numeros = [1, 2, 3, 4, 5]
print(numeros)
```

Ejercicio 2

Enunciado: Crea una lista de frutas y agrega una fruta más al final.
Explicación: Se aprende a usar `append()` para añadir elementos al final de la lista.

```
frutas = ["manzana", "plátano", "naranja"]
frutas.append("pera")
print(frutas)
```

Ejercicio 3

Enunciado: Crea una lista de 5 números y muestra su longitud.
Explicación: Se usa la función `len()` para obtener el tamaño de una lista.

```
numeros = [10, 20, 30, 40, 50]
print(len(numeros))
```

Ejercicio 4

Enunciado: Muestra el primer y último elemento de una lista de nombres.
Explicación: Se aprende a acceder a elementos de la lista mediante índices.

```
nombres = ["Ana", "Luis", "Carlos", "Marta"]
print("Primero:", nombres[0])
print("Último:", nombres[-1])
```

Ejercicio 5

Enunciado: Elimina un elemento de la lista de colores por su valor.
Explicación: Se utiliza `remove()` para borrar un elemento específico.

```
colores = ["rojo", "verde", "azul", "amarillo"]
colores.remove("verde")
print(colores)
```

Ejercicio 6

Enunciado: Inserta un número en la segunda posición de la lista.
Explicación: Se usa `insert(posición, valor)` para añadir un elemento en un índice específico.

```
numeros = [1, 3, 4, 5]
numeros.insert(1, 2)
print(numeros)
```

Ejercicio 7

Enunciado: Ordena una lista de números de menor a mayor.
Explicación: Se aprende a usar `sort()` para ordenar listas numéricas.

```
numeros = [5, 2, 8, 1, 4]
numeros.sort()
print(numeros)
```

Ejercicio 8

Enunciado: Invierte el orden de los elementos de una lista.
Explicación: Se utiliza `reverse()` para invertir la lista.

```
numeros = [1, 2, 3, 4, 5]
numeros.reverse()
print(numeros)
```

Ejercicio 9

Enunciado: Cuenta cuántas veces aparece un número en la lista.
Explicación: `count()` permite contar la frecuencia de un elemento en la lista.

```
numeros = [1, 2, 2, 3, 2, 4]
print(numeros.count(2))
```

Ejercicio 10

Enunciado: Copia una lista en otra variable y muéstrala.
Explicación: `copy()` permite crear una copia independiente de la lista original.

```
original = [1, 2, 3]
copia = original.copy()
print(copia)
```

Ejercicio 11

Enunciado: Suma todos los números de una lista y muestra el resultado.
Explicación: Se aprende a recorrer la lista con un bucle `for` y sumar sus elementos.

```
numeros = [1, 2, 3, 4, 5]

suma = 0

for num in numeros:

    suma += num

print(suma)
```

Ejercicio 12

Enunciado: Encuentra el número máximo en una lista.
Explicación: La función `max()` devuelve el valor más grande de la lista.

```
numeros = [10, 3, 7, 21, 5]

print(max(numeros))
```

Ejercicio 13

Enunciado: Encuentra el número mínimo en una lista.
Explicación: La función `min()` devuelve el valor más pequeño de la lista.

```
numeros = [10, 3, 7, 21, 5]

print(min(numeros))
```

Ejercicio 14

Enunciado: Multiplica cada elemento de una lista por 2 y guarda los resultados en otra lista.
Explicación: Se utiliza un bucle `for` para procesar cada elemento y crear una lista nueva.

```
numeros = [1, 2, 3, 4, 5]

dobles = []

for num in numeros:

    dobles.append(num * 2)

print(dobles)
```

Ejercicio 15

Enunciado: Verifica si un elemento existe en una lista.
Explicación: Se utiliza el operador `in` para comprobar la existencia de un elemento.

```
colores = ["rojo", "verde", "azul"]
print("verde" in colores)
print("amarillo" in colores)
```

Ejercicio 16

Enunciado: Elimina el último elemento de una lista.
Explicación: `pop()` sin parámetros elimina y devuelve el último elemento de la lista.

```
numeros = [1, 2, 3, 4, 5]
numeros.pop()
print(numeros)
```

Ejercicio 17

Enunciado: Muestra todos los elementos de una lista usando un bucle `for`.
Explicación: Se aprende a recorrer la lista para imprimir cada elemento individualmente.

```
frutas = ["manzana", "plátano", "naranja"]
for fruta in frutas:
    print(fruta)
```

Ejercicio 18

Enunciado: Une dos listas en una sola.
Explicación: Se utiliza el operador `+` para concatenar listas.

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
lista_completa = lista1 + lista2
print(lista_completa)
```

Ejercicio 19

Enunciado: Repite los elementos de una lista 3 veces.
Explicación: Se utiliza el operador `*` para repetir listas.

```
numeros = [1, 2, 3]
repetidos = numeros * 3
print(repetidos)
```

Ejercicio 20

Enunciado: Convierte una cadena de texto en una lista de caracteres.
Explicación: La función `list()` permite transformar un `str` en una lista de sus caracteres individuales.

```
palabra = "Python"

letras = list(palabra)

print(letras)
```

Ejercicio 21

Enunciado: Muestra el índice de un elemento en la lista.
Explicación: Se utiliza `index()` para obtener la posición de un elemento.

```
colores = ["rojo", "verde", "azul"]

print(colores.index("verde"))
```

Ejercicio 22

Enunciado: Convierte una lista de números en una lista de cadenas (strings).
Explicación: Se recorre la lista y se transforma cada elemento con `str()`.

```
numeros = [1, 2, 3, 4, 5]

cadenas = [str(num) for num in numeros]

print(cadenas)
```

Ejercicio 23

Enunciado: Elimina todos los elementos de una lista.
Explicación: `clear()` vacía completamente la lista.

```
numeros = [1, 2, 3, 4, 5]

numeros.clear()

print(numeros)
```

Ejercicio 24

Enunciado: Imprime solo los elementos pares de una lista.
Explicación: Se usa un bucle `for` con condición `if` para filtrar elementos pares.

```
numeros = [1, 2, 3, 4, 5, 6]

for num in numeros:

    if num % 2 == 0:

        print(num)
```

Ejercicio 25

Enunciado: Crea una lista con números del 1 al 10 usando `range()`.
Explicación: `list(range())` genera listas de números consecutivos.

```
numeros = list(range(1, 11))
print(numeros)
```

Ejercicio 26

Enunciado: Cuenta cuántos elementos hay en una lista que sean mayores que 5.
Explicación: Se recorre la lista y se incrementa un contador según la condición.

```
numeros = [3, 6, 8, 2, 10, 5]
contador = 0
for num in numeros:
    if num > 5:
        contador += 1
print(contador)
```

Ejercicio 27

Enunciado: Crea una lista de nombres y ordénala alfabéticamente.
Explicación: `sort()` funciona también con cadenas, ordenándolas según el orden alfabético.

```
nombres = ["Luis", "Ana", "Carlos", "Marta"]
nombres.sort()
print(nombres)
```

Ejercicio 28

Enunciado: Convierte una lista de números en una lista con sus cuadrados.
Explicación: Se recorre la lista y se eleva cada elemento al cuadrado.

```
numeros = [1, 2, 3, 4, 5]
cuadrados = [num ** 2 for num in numeros]
print(cuadrados)
```

Ejercicio 29

Enunciado: Muestra la lista en formato de cadena separada por comas.
Explicación: `join()` convierte listas de cadenas en un solo string separado por un delimitador.

```
frutas = ["manzana", "plátano", "naranja"]
resultado = ", ".join(frutas)
print(resultado)
```

Ejercicio 30

Enunciado: Copia los elementos de una lista a otra usando un bucle `for`.
Explicación: Se aprende a crear una copia manualmente recorriendo los elementos.

```
original = [1, 2, 3, 4, 5]
copia = []
for num in original:
    copia.append(num)
print(copia)
```

Ejercicio 31

Enunciado: Muestra los elementos de una lista en orden inverso usando índices.
Explicación: Se recorre la lista usando índices negativos para imprimirla al revés.

```
numeros = [1, 2, 3, 4, 5]
for i in range(len(numeros)-1, -1, -1):
    print(numeros[i])
```

Ejercicio 32

Enunciado: Intercambia el primer y último elemento de una lista.
Explicación: Se aprende a modificar elementos mediante índices.

```
numeros = [1, 2, 3, 4, 5]
numeros[0], numeros[-1] = numeros[-1], numeros[0]
print(numeros)
```

Ejercicio 33

Enunciado: Muestra los elementos de una lista que estén en posiciones pares.
Explicación: Se recorren los índices de la lista y se filtran los pares.

```
numeros = [10, 20, 30, 40, 50]
for i in range(len(numeros)):
    if i % 2 == 0:
        print(numeros[i])
```

Ejercicio 34

Enunciado: Crea una lista con los primeros 10 números impares.
Explicación: Se utiliza `range()` con paso 2 para generar los impares.

```
impares = list(range(1, 20, 2))
print(impares)
```

Ejercicio 35

Enunciado: Dada una lista de nombres, imprime los que tienen más de 4 letras.
Explicación: Se filtran los elementos según su longitud usando `len()`.

```
nombres = ["Ana", "Carlos", "Luis", "Marta"]
for nombre in nombres:
    if len(nombre) > 4:
        print(nombre)
```

Ejercicio 36

Enunciado: Convierte todos los elementos de una lista de cadenas a mayúsculas.
Explicación: Se recorre la lista y se aplica `upper()` a cada elemento.

```
frutas = ["manzana", "plátano", "naranja"]
mayusculas = [fruta.upper() for fruta in frutas]
print(mayusculas)
```

Ejercicio 37

Enunciado: Elimina todos los elementos que sean menores que 5 en una lista.
Explicación: Se recorre la lista y se construye una nueva con los elementos que cumplen la condición.

```
numeros = [2, 5, 8, 1, 7, 3]
numeros_filtrados = [num for num in numeros if num >= 5]
print(numeros_filtrados)
```

Ejercicio 38

Enunciado: Suma elemento a elemento dos listas de igual tamaño.
Explicación: Se recorren las listas usando índices y se suman los elementos correspondientes.

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
suma = []
for i in range(len(lista1)):
    suma.append(lista1[i] + lista2[i])
print(suma)
```

Ejercicio 39

Enunciado: Crea una lista que contenga los caracteres de otra lista pero sin repeticiones.
Explicación: Se usa un bucle y se agrega un elemento solo si no está ya en la nueva lista.

```
letras = ["a", "b", "a", "c", "b"]
sin_repetir = []
for letra in letras:
    if letra not in sin_repetir:
        sin_repetir.append(letra)
print(sin_repetir)
```

Ejercicio 40

Enunciado: Muestra los elementos de una lista multiplicados por su índice.
Explicación: Se recorre la lista con `enumerate()` para obtener índice y valor.

```
numeros = [1, 2, 3, 4, 5]
for i, num in enumerate(numeros):
    print(num * i)
```

Ejercicio 41

Enunciado: Crea una lista con los primeros 10 números cuadrados.
Explicación: Se genera una lista usando comprensión y se eleva cada número al cuadrado.

```
cuadrados = [x**2 for x in range(1, 11)]
print(cuadrados)
```

Ejercicio 42

Enunciado: Dada una lista de números, muestra solo los números positivos.
Explicación: Se filtran los elementos con un bucle y una condición `if`.

```
numeros = [-2, 5, -1, 8, 0, -3]
positivos = [num for num in numeros if num > 0]
print(positivos)
```

Ejercicio 43

Enunciado: Crea una lista con los elementos de otra lista multiplicados por 3.
Explicación: Se recorre la lista original y se multiplica cada elemento.

```
numeros = [1, 2, 3, 4, 5]
triplicados = [num * 3 for num in numeros]
print(triplicados)
```

Ejercicio 44

Enunciado: Cuenta cuántos elementos de una lista son cadenas de texto.
Explicación: Se usa `isinstance()` para verificar el tipo de cada elemento.

```
elementos = [1, "hola", 3, "mundo", 5]
contador = 0
for elem in elementos:
    if isinstance(elem, str):
        contador += 1
print(contador)
```

Ejercicio 45

Enunciado: Dada una lista de números, crea otra con los números al cuadrado solo si son pares.
Explicación: Se combina comprensión de listas con condición `if`.

```
numeros = [1, 2, 3, 4, 5, 6]
cuadrados_pares = [num**2 for num in numeros if num % 2 == 0]
print(cuadrados_pares)
```

Ejercicio 46

Enunciado: Intercambia los elementos de una lista usando un bucle: el primero con el último, el segundo con el penúltimo, etc.
Explicación: Se recorre hasta la mitad de la lista y se intercambian los elementos.

```
numeros = [1, 2, 3, 4, 5, 6]
for i in range(len(numeros)//2):
    numeros[i], numeros[-i-1] = numeros[-i-1], numeros[i]
print(numeros)
```

Ejercicio 47

Enunciado: Crea una lista que contenga solo los elementos únicos de otra lista.
Explicación: Se puede usar un bucle y verificar que el elemento no esté ya en la nueva lista.

```
numeros = [1, 2, 2, 3, 4, 4, 5]
unicos = []
for num in numeros:
    if num not in unicos:
        unicos.append(num)
print(unicos)
```

Ejercicio 48

Enunciado: Une dos listas de igual tamaño sumando sus elementos correspondientes.
Explicación: Se usa un bucle `for` con índice para sumar los elementos de la misma posición.

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
suma = []
for i in range(len(lista1)):
    suma.append(lista1[i] + lista2[i])
print(suma)
```

Ejercicio 49

Enunciado: Dada una lista de cadenas, crea otra lista con la longitud de cada cadena.
Explicación: Se recorre la lista y se aplica `len()` a cada elemento.

```
palabras = ["hola", "mundo", "python"]
longitudes = [len(palabra) for palabra in palabras]
print(longitudes)
```

Ejercicio 50

Enunciado: Convierte una lista de números en una cadena separada por guiones.
Explicación: Primero se convierten los números en cadenas y luego se usa `join()`.

```
numeros = [1, 2, 3, 4, 5]
resultado = "-".join([str(num) for num in numeros])
print(resultado)
```

Ejercicio 51

Enunciado: Dada una lista de números, muestra los números mayores que el promedio de la lista.
Explicación: Primero se calcula el promedio y luego se filtran los elementos mayores que él.

```
numeros = [2, 5, 8, 3, 10]
promedio = sum(numeros) / len(numeros)
mayores = [num for num in numeros if num > promedio]
print(mayores)
```

Ejercicio 52

Enunciado: Crea una lista con los elementos de otra lista repetidos dos veces.
Explicación: Se usa el operador `*` para repetir la lista.

```
numeros = [1, 2, 3]
repetidos = numeros * 2
print(repetidos)
```

Ejercicio 53

Enunciado: Crea una lista con los elementos de otra lista en orden inverso usando slicing.
Explicación: Se utiliza `[::-1]` para invertir la lista de forma sencilla.

```
numeros = [1, 2, 3, 4, 5]
invertidos = numeros[::-1]
print(invertidos)
```

Ejercicio 54

Enunciado: Dada una lista de números, reemplaza los negativos por cero.
Explicación: Se recorre la lista y se modifica cada elemento según su valor.

```
numeros = [2, -5, 8, -3, 0]
for i in range(len(numeros)):
    if numeros[i] < 0:
        numeros[i] = 0
print(numeros)
```

Ejercicio 55

Enunciado: Crea una lista de los números del 1 al 20 que sean divisibles entre 3.
Explicación: Se filtran los números usando `range()` y condición `if`.

```
div3 = [x for x in range(1, 21) if x % 3 == 0]
print(div3)
```

Ejercicio 56

Enunciado: Dada una lista de cadenas, elimina las que contengan la letra “a”.
Explicación: Se construye una nueva lista filtrando con condición `if "a" not in cadena`.

```
palabras = ["hola", "mundo", "python", "casa"]
filtradas = [palabra for palabra in palabras if "a" not in palabra]
print(filtradas)
```

Ejercicio 57

Enunciado: Dada una lista de números, crea otra lista con los números multiplicados por su índice.
Explicación: Se recorre la lista con `enumerate()` para obtener índice y valor.

```
numeros = [1, 2, 3, 4, 5]

resultado = [num * i for i, num in enumerate(numeros)]

print(resultado)
```

Ejercicio 58

Enunciado: Dada una lista de palabras, convierte la primera letra de cada palabra en mayúscula.
Explicación: Se recorre la lista y se aplica `capitalize()` a cada elemento.

```
palabras = ["hola", "mundo", "python"]

capitalizadas = [palabra.capitalize() for palabra in palabras]

print(capitalizadas)
```

Ejercicio 59

Enunciado: Dada una lista de números, separa los pares de los impares en dos listas distintas.
Explicación: Se recorren los elementos y se agregan a la lista correspondiente según paridad.

```
numeros = [1, 2, 3, 4, 5, 6]

pares = [num for num in numeros if num % 2 == 0]

impares = [num for num in numeros if num % 2 != 0]

print("Pares:", pares)

print("Impares:", impares)
```

Ejercicio 60

Enunciado: Dada una lista de cadenas, cuenta cuántas empiezan con la letra “p”.
Explicación: Se recorre la lista y se verifica con `startswith()`.

```
palabras = ["python", "programa", "hola", "prueba"]

contador = 0

for palabra in palabras:

    if palabra.startswith("p"):

        contador += 1

print(contador)
```

Ejercicio 61

Enunciado: Dada una lista de números, muestra el segundo elemento más grande.
Explicación: Se ordena la lista de mayor a menor y se accede al segundo elemento.

```
numeros = [10, 3, 7, 21, 5]
numeros.sort(reverse=True)
print(numeros[1])
```

Ejercicio 62

Enunciado: Dada una lista de palabras, crea una lista con la longitud de cada palabra.
Explicación: Se usa comprensión de listas y `len()` para obtener la longitud de cada cadena.

```
palabras = ["hola", "mundo", "python"]
longitudes = [len(p) for p in palabras]
print(longitudes)
```

Ejercicio 63

Enunciado: Dada una lista de números, crea otra lista con los números que sean múltiplos de 5.
Explicación: Se recorre la lista y se filtran los elementos usando condición `if`.

```
numeros = [5, 12, 15, 22, 30]
mult5 = [num for num in numeros if num % 5 == 0]
print(mult5)
```

Ejercicio 64

Enunciado: Dada una lista de palabras, convierte todas a minúsculas.
Explicación: Se recorre la lista y se aplica `lower()` a cada elemento.

```
palabras = ["HOLA", "Mundo", "Python"]
minusculas = [p.lower() for p in palabras]
print(minusculas)
```

Ejercicio 65

Enunciado: Suma todos los números impares de una lista.
Explicación: Se recorre la lista y se suman solo los números impares usando `if num % 2 != 0`.

```
numeros = [1, 2, 3, 4, 5, 6]
suma_impares = sum(num for num in numeros if num % 2 != 0)
print(suma_impares)
```

Ejercicio 66

Enunciado: Dada una lista de palabras, crea otra lista con las palabras que tengan más de 3 letras.
Explicación: Se usa comprensión de listas y `len()` para filtrar las palabras.

```
palabras = ["sol", "luna", "cielo", "mar"]
filtradas = [p for p in palabras if len(p) > 3]
print(filtradas)
```

Ejercicio 67

Enunciado: Dada una lista de números, reemplaza los números mayores que 10 por 10.
Explicación: Se recorre la lista y se modifica el valor según la condición.

```
numeros = [5, 12, 8, 15, 3]
for i in range(len(numeros)):
    if numeros[i] > 10:
        numeros[i] = 10
print(numeros)
```

Ejercicio 68

Enunciado: Dada una lista de números, crea otra lista con los números negativos convertidos en positivos.
Explicación: Se usa `abs()` para convertir los números negativos en positivos.

```
numeros = [-2, 5, -7, 8, -3]
positivos = [abs(num) for num in numeros]
print(positivos)
```

Ejercicio 69

Enunciado: Dada una lista de cadenas, elimina las cadenas vacías.
Explicación: Se construye una nueva lista filtrando las cadenas que no estén vacías.

```
palabras = ["hola", "", "python", "", "mundo"]
no_vacias = [p for p in palabras if p != ""]
print(no_vacias)
```

Ejercicio 70

Enunciado: Dada una lista de números, crea otra lista con los números al cuadrado solo si son mayores que 5.
Explicación: Se combina comprensión de listas con condición `if` para filtrar y elevar al cuadrado.

```
numeros = [2, 5, 8, 3, 10]

cuadrados_mayores5 = [num**2 for num in numeros if num > 5]

print(cuadrados_mayores5)
```

Ejercicio 71

Enunciado: Dada una lista de números, muestra los elementos que son múltiplos de 2 o 3.
Explicación: Se filtran los elementos con condición combinada usando `if num % 2 == 0 or num % 3 == 0`.

```
numeros = [1, 2, 3, 4, 5, 6, 7, 9]

mult2o3 = [num for num in numeros if num % 2 == 0 or num % 3 == 0]

print(mult2o3)
```

Ejercicio 72

Enunciado: Dada una lista de palabras, crea otra lista con las palabras que contienen la letra “o”.
Explicación: Se recorre la lista y se filtran las palabras usando `if "o" in palabra`.

```
palabras = ["hola", "mundo", "python", "cielo"]

contienen_o = [p for p in palabras if "o" in p]

print(contienen_o)
```

Ejercicio 73

Enunciado: Dada una lista de números, duplica cada elemento usando comprensión de listas.
Explicación: Se multiplica cada número por 2 dentro de la comprensión de listas.

```
numeros = [1, 2, 3, 4, 5]

duplicados = [num * 2 for num in numeros]

print(duplicados)
```

Ejercicio 74

Enunciado: Dada una lista de palabras, crea otra lista con las palabras en orden inverso (carácter a carácter).
Explicación: Se recorre cada palabra y se invierte usando slicing `[::-1]`.

```
palabras = ["hola", "mundo", "python"]

invertidas = [p[::-1] for p in palabras]

print(invertidas)
```

Ejercicio 75

Enunciado: Dada una lista de números, elimina los duplicados sin usar sets.
Explicación: Se recorre la lista y se agrega a una nueva lista solo si no está ya presente.

```
numeros = [1, 2, 2, 3, 4, 4, 5]

sin_duplicados = []

for num in numeros:

    if num not in sin_duplicados:

        sin_duplicados.append(num)

print(sin_duplicados)
```

Ejercicio 76

Enunciado: Dada una lista de palabras, muestra las palabras que tengan exactamente 5 letras.
Explicación: Se filtran las palabras usando `len() == 5`.

```
palabras = ["hola", "mundo", "python", "luz", "perro"]

exactamente5 = [p for p in palabras if len(p) == 5]

print(exactamente5)
```

Ejercicio 77

Enunciado: Dada una lista de números, reemplaza los ceros por la media de la lista.
Explicación: Primero se calcula la media de los números distintos de cero y luego se reemplazan los ceros.

```
numeros = [0, 2, 0, 8, 10]

media = sum(num for num in numeros if num != 0) / len([num for num in numeros if num != 0])

for i in range(len(numeros)):

    if numeros[i] == 0:

        numeros[i] = media

print(numeros)
```

Ejercicio 78

Enunciado: Dada una lista de palabras, crea una lista con la primera letra de cada palabra.
Explicación: Se recorre la lista y se toma el primer carácter de cada cadena.

```
palabras = ["hola", "mundo", "python"]

primeras_letras = [p[0] for p in palabras]

print(primeras_letras)
```

Ejercicio 79

Enunciado: Dada una lista de números, cuenta cuántos son negativos.
Explicación: Se recorre la lista y se incrementa un contador para cada número menor que 0.

```
numeros = [-2, 5, -7, 8, -3]

contador = sum(1 for num in numeros if num < 0)

print(contador)
```

Ejercicio 80

Enunciado: Dada una lista de palabras, muestra la palabra más larga.
Explicación: Se usa `max()` con la función `key=len` para obtener la cadena de mayor longitud.

```
palabras = ["hola", "mundo", "python", "programación"]

mas_larga = max(palabras, key=len)

print(mas_larga)
```

Ejercicio 81

Enunciado: Dada una lista de números, crea otra lista con los números divididos entre 2.
Explicación: Se recorre la lista y se divide cada elemento usando comprensión de listas.

```
numeros = [2, 4, 6, 8, 10]

divididos = [num / 2 for num in numeros]

print(divididos)
```

Ejercicio 82

Enunciado: Dada una lista de palabras, elimina las palabras que contengan más de 4 letras.
Explicación: Se filtran los elementos usando `len() <= 4`.

```
palabras = ["sol", "luna", "estrella", "mar"]

cortas = [p for p in palabras if len(p) <= 4]

print(cortas)
```

Ejercicio 83

Enunciado: Dada una lista de números, reemplaza los números impares por cero.
Explicación: Se recorre la lista y se modifica cada elemento según la condición `num % 2 != 0`.

```
numeros = [1, 2, 3, 4, 5, 6]

for i in range(len(numeros)):

    if numeros[i] % 2 != 0:

        numeros[i] = 0

print(numeros)
```

Ejercicio 84

Enunciado: Dada una lista de palabras, convierte las palabras que empiecen con “p” a mayúsculas.
Explicación: Se recorre la lista y se aplica `upper()` a las palabras que cumplan la condición.

```
palabras = ["python", "programa", "hola", "prueba"]
modificadas = [p.upper() if p.startswith("p") else p for p in palabras]
print(modificadas)
```

Ejercicio 85

Enunciado: Dada una lista de números, muestra solo los números menores que el promedio.
Explicación: Se calcula el promedio y se filtran los números que sean menores.

```
numeros = [2, 5, 8, 3, 10]
promedio = sum(numeros) / len(numeros)
menores = [num for num in numeros if num < promedio]
print(menores)
```

Ejercicio 86

Enunciado: Dada una lista de palabras, crea una lista con las palabras al revés y en mayúsculas.
Explicación: Se combinan slicing `[::-1]` y `upper()` en comprensión de listas.

```
palabras = ["hola", "mundo", "python"]
modificadas = [p[::-1].upper() for p in palabras]
print(modificadas)
```

Ejercicio 87

Enunciado: Dada una lista de números, intercambia los elementos pares con los impares (índices pares con índices impares).
Explicación: Se recorre la lista de dos en dos y se intercambian los elementos consecutivos.

```
numeros = [1, 2, 3, 4, 5, 6]
for i in range(0, len(numeros)-1, 2):
    numeros[i], numeros[i+1] = numeros[i+1], numeros[i]
print(numeros)
```

Ejercicio 88

Enunciado: Dada una lista de palabras, crea otra lista con la longitud de cada palabra multiplicada por 2.
Explicación: Se recorre la lista y se multiplica la longitud usando `len() * 2`.

```
palabras = ["hola", "mundo", "python"]

longitudes_dobles = [len(p) * 2 for p in palabras]

print(longitudes_dobles)
```

Ejercicio 89

Enunciado: Dada una lista de números, crea otra lista con los elementos positivos primero y los negativos después.
Explicación: Se usan dos comprensiones de listas y se concatenan.

```
numeros = [3, -2, 5, -1, 0, -4]

ordenados = [num for num in numeros if num >= 0] + [num for num in numeros if num < 0]

print(ordenados)
```

Ejercicio 90

Enunciado: Dada una lista de palabras, muestra la palabra más corta.
Explicación: Se usa `min()` con `key=len` para obtener la palabra de menor longitud.

```
palabras = ["hola", "mundo", "py", "python"]

mas_corta = min(palabras, key=len)

print(mas_corta)
```

Ejercicio 91

Enunciado: Dada una lista de números, crea otra lista con los elementos que no sean múltiplos de 3.
Explicación: Se filtran los elementos usando `if num % 3 != 0`.

```
numeros = [1, 2, 3, 4, 5, 6, 7, 9]
sin_mult3 = [num for num in numeros if num % 3 != 0]
print(sin_mult3)
```

Ejercicio 92

Enunciado: Dada una lista de palabras, reemplaza las palabras que tengan más de 5 letras por “larga”.

Explicación: Se recorre la lista y se usa una condición para modificar las palabras.

```
palabras = ["hola", "mundo", "python", "programación"]
modificadas = [p if len(p) <= 5 else "larga" for p in palabras]
print(modificadas)
```

Ejercicio 93

Enunciado: Dada una lista de números, muestra el producto de todos sus elementos.

Explicación: Se recorre la lista multiplicando los elementos para obtener el producto total.

```
numeros = [1, 2, 3, 4, 5]
producto = 1
for num in numeros:
    producto *= num
print(producto)
```

Ejercicio 94

Enunciado: Dada una lista de palabras, crea otra lista con las palabras que terminen en “o”.

Explicación: Se filtran las palabras usando `endswith("o")`.

```
palabras = ["hola", "mundo", "python", "programo"]
terminan_o = [p for p in palabras if p.endswith("o")]
print(terminan_o)
```

Ejercicio 95

Enunciado: Dada una lista de números, reemplaza los números mayores que 10 por su mitad.

Explicación: Se recorre la lista y se divide el número por 2 si es mayor que 10.

```
numeros = [5, 12, 8, 15, 3]
for i in range(len(numeros)):
    if numeros[i] > 10:
        numeros[i] /= 2
print(numeros)
```

Ejercicio 96

Enunciado: Dada una lista de palabras, crea otra lista con las palabras duplicadas (concatenadas consigo mismas).

Explicación: Se recorre la lista y se concatena cada palabra consigo misma usando `+`.

```
palabras = ["hola", "mundo", "python"]
duplicadas = [p + p for p in palabras]
print(duplicadas)
```

Ejercicio 97

Enunciado: Dada una lista de números, muestra la suma de los elementos en posiciones impares.
Explicación: Se recorre la lista usando índices y se suman los elementos de índices impares.

```
numeros = [1, 2, 3, 4, 5, 6]
suma_impares = sum(numeros[i] for i in range(1, len(numeros), 2))
print(suma_impares)
```

Ejercicio 98

Enunciado: Dada una lista de palabras, crea otra lista con las palabras invertidas.
Explicación: Se usa slicing `[::-1]` para invertir cada palabra.

```
palabras = ["hola", "mundo", "python"]
invertidas = [p[::-1] for p in palabras]
print(invertidas)
```

Ejercicio 99

Enunciado: Dada una lista de números, crea otra lista con los números elevados al cubo.
Explicación: Se recorre la lista y se aplica `num**3` a cada elemento.

```
numeros = [1, 2, 3, 4, 5]
cubos = [num**3 for num in numeros]
print(cubos)
```

Ejercicio 100

Enunciado: Dada una lista de palabras, muestra las palabras ordenadas por longitud, de menor a mayor.
Explicación: Se usa `sorted()` con `key=len` para ordenar según la longitud de cada palabra.

```
palabras = ["python", "hola", "mundo", "py"]
ordenadas = sorted(palabras, key=len)
print(ordenadas)
```

Ejercicios python conjuntos

Ejercicio 1

Enunciado: Crea un conjunto con los números del 1 al 5 y muéstralo.
Explicación: Un `set` almacena elementos únicos sin orden específico.

```
numeros = {1, 2, 3, 4, 5}
print(numeros)
```

Ejercicio 2

Enunciado: Crea un conjunto a partir de una lista que contiene números repetidos.
Explicación: `set()` elimina automáticamente los elementos duplicados.

```
lista = [1, 2, 2, 3, 4, 4, 5]
conjunto = set(lista)
print(conjunto)
```

Ejercicio 3

Enunciado: Añade un nuevo elemento a un conjunto.
Explicación: `add()` permite agregar un elemento a un conjunto existente.

```
numeros = {1, 2, 3}
numeros.add(4)
print(numeros)
```

Ejercicio 4

Enunciado: Elimina un elemento de un conjunto.
Explicación: `remove()` elimina un elemento, pero da error si no existe.

```
numeros = {1, 2, 3, 4}
numeros.remove(3)
print(numeros)
```

Ejercicio 5

Enunciado: Elimina un elemento de un conjunto sin que ocurra error si no existe.
Explicación: `discard()` elimina un elemento si está, y no hace nada si no lo está.

```
numeros = {1, 2, 3, 4}
numeros.discard(5)
print(numeros)
```

Ejercicio 6

Enunciado: Elimina todos los elementos de un conjunto.
Explicación: `clear()` deja el conjunto vacío.

```
numeros = {1, 2, 3, 4}
numeros.clear()
print(numeros)
```

Ejercicio 7

Enunciado: Comprueba si un elemento pertenece a un conjunto.
Explicación: `in` devuelve `True` si el elemento está en el conjunto, `False` si no.

```
numeros = {1, 2, 3, 4}
print(3 in numeros)
print(5 in numeros)
```

Ejercicio 8

Enunciado: Une dos conjuntos en uno solo.
Explicación: `union()` o el operador `|` combinan todos los elementos de ambos conjuntos sin duplicados.

```
a = {1, 2, 3}
b = {3, 4, 5}
c = a.union(b)
print(c)
```

Ejercicio 9

Enunciado: Muestra la intersección de dos conjuntos.
Explicación: `intersection()` o `&` devuelven los elementos comunes a ambos conjuntos.

```
a = {1, 2, 3}
b = {2, 3, 4}
inter = a.intersection(b)
print(inter)
```

Ejercicio 10

Enunciado: Muestra la diferencia de dos conjuntos.
Explicación: `difference()` o `-` devuelven los elementos que están en el primer conjunto y no en el segundo.

```
a = {1, 2, 3}
b = {2, 3, 4}
diff = a.difference(b)
print(diff)
```

Ejercicio 11

Enunciado: Muestra la diferencia simétrica entre dos conjuntos.
Explicación: `symmetric_difference()` o `^` devuelve los elementos que están en uno u otro conjunto, pero no en ambos.

```
a = {1, 2, 3}
b = {3, 4, 5}
diff_sim = a.symmetric_difference(b)
print(diff_sim)
```

Ejercicio 12

Enunciado: Comprueba si un conjunto es subconjunto de otro.
Explicación: `issubset()` devuelve `True` si todos los elementos del primer conjunto están en el segundo.

```
a = {1, 2}
b = {1, 2, 3, 4}
print(a.issubset(b))
```

Ejercicio 13

Enunciado: Comprueba si un conjunto es superconjunto de otro.
Explicación: `issuperset()` devuelve `True` si todos los elementos del segundo conjunto están en el primero.

```
a = {1, 2, 3, 4}

b = {2, 3}

print(a.issuperset(b))
```

Ejercicio 14

Enunciado: Copia un conjunto en otro conjunto.
Explicación: `copy()` crea una copia independiente del conjunto original.

```
original = {1, 2, 3}

copia = original.copy()

print(copia)
```

Ejercicio 15

Enunciado: Elimina un elemento aleatorio de un conjunto.
Explicación: `pop()` elimina y devuelve un elemento arbitrario, ya que los conjuntos no tienen orden.

```
numeros = {1, 2, 3, 4}

elemento = numeros.pop()

print(elemento)

print(numeros)
```

Ejercicio 16

Enunciado: Cuenta cuántos elementos hay en un conjunto.
Explicación: `len()` devuelve el número de elementos únicos en el conjunto.

```
numeros = {1, 2, 3, 4, 5}

print(len(numeros))
```

Ejercicio 17

Enunciado: Convierte una cadena en un conjunto de caracteres.
Explicación: `set()` puede usarse para obtener todos los caracteres únicos de una cadena.

```
texto = "python"

conjunto_caracteres = set(texto)

print(conjunto_caracteres)
```

Ejercicio 18

Enunciado: Convierte una lista con elementos repetidos en un conjunto para eliminar duplicados.
Explicación: `set()` elimina automáticamente los duplicados de la lista.

```
lista = [1, 2, 2, 3, 4, 4, 5]
conjunto = set(lista)
print(conjunto)
```

Ejercicio 19

Enunciado: Une varios conjuntos a la vez usando `union()`.
Explicación: `union()` puede recibir múltiples conjuntos y combinar todos los elementos únicos.

```
a = {1, 2}
b = {2, 3}
c = {3, 4}
resultado = a.union(b, c)
print(resultado)
```

Ejercicio 20

Enunciado: Interseca varios conjuntos a la vez usando `intersection()`.
Explicación: `intersection()` devuelve los elementos comunes a todos los conjuntos.

```
a = {1, 2, 3}
b = {2, 3, 4}
c = {3, 5, 6}
comun = a.intersection(b, c)
print(comun)
```

Ejercicio 21

Enunciado: Elimina todos los elementos de un conjunto que estén en otro conjunto.
Explicación: `difference_update()` elimina del conjunto original todos los elementos que existen en otro conjunto.

```
a = {1, 2, 3, 4}
b = {2, 4}
a.difference_update(b)
print(a)
```

Ejercicio 22

Enunciado: Mantén solo los elementos que están en ambos conjuntos.
Explicación: `intersection_update()` modifica el conjunto original conservando solo los elementos comunes.

```
a = {1, 2, 3, 4}
b = {3, 4, 5}
a.intersection_update(b)
print(a)
```

Ejercicio 23

Enunciado: Elimina los elementos de un conjunto que también estén en otro conjunto (diferencia simétrica).
Explicación: `symmetric_difference_update()` actualiza el conjunto con elementos que están en uno u otro conjunto, pero no en ambos.

```
a = {1, 2, 3}
b = {3, 4, 5}
a.symmetric_difference_update(b)
print(a)
```

Ejercicio 24

Enunciado: Comprueba si dos conjuntos no tienen elementos en común.
Explicación: `isdisjoint()` devuelve `True` si no hay ningún elemento compartido.

```
a = {1, 2, 3}
b = {4, 5, 6}
print(a.isdisjoint(b))
```

Ejercicio 25

Enunciado: Convierte un conjunto en una lista.
Explicación: `list()` permite transformar un conjunto en una lista para poder usar índices.

```
conjunto = {1, 2, 3, 4}
lista = list(conjunto)
print(lista)
```

Ejercicio 26

Enunciado: Convierte un conjunto en una tupla.
Explicación: `tuple()` permite transformar un conjunto en una tupla para tener elementos inmutables.

```
conjunto = {1, 2, 3, 4}

tupla = tuple(conjunto)

print(tupla)
```

Ejercicio 27

Enunciado: Itera sobre los elementos de un conjunto y muéstralos.
Explicación: Un conjunto se puede recorrer con un bucle `for`, aunque no tiene orden.

```
conjunto = {1, 2, 3, 4}

for num in conjunto:

    print(num)
```

Ejercicio 28

Enunciado: Crea un conjunto con números pares del 2 al 10 usando `range()`.
Explicación: `set()` puede recibir un rango para generar elementos automáticamente.

```
pares = set(range(2, 11, 2))

print(pares)
```

Ejercicio 29

Enunciado: Añade varios elementos a un conjunto a la vez.
Explicación: `update()` permite agregar múltiples elementos de una lista, tupla o conjunto al conjunto original.

```
conjunto = {1, 2}

conjunto.update([3, 4, 5])

print(conjunto)
```

Ejercicio 30

Enunciado: Elimina elementos de un conjunto usando un bucle y condición.
Explicación: Se recorre una copia del conjunto para eliminar elementos del original según una condición.

```
numeros = {1, 2, 3, 4, 5}

for num in numeros.copy():

    if num % 2 == 0:

        numeros.remove(num)

print(numeros)
```

Ejercicio 31

Enunciado: Dada una lista de palabras, crea un conjunto con todas las palabras únicas.
Explicación: `set()` elimina duplicados y mantiene solo elementos únicos.

```
palabras = ["hola", "mundo", "hola", "python"]

conjunto = set(palabras)

print(conjunto)
```

Ejercicio 32

Enunciado: Dado un conjunto de números, calcula su suma.
Explicación: `sum()` suma todos los elementos únicos del conjunto.

```
numeros = {1, 2, 3, 4, 5}

total = sum(numeros)

print(total)
```

Ejercicio 33

Enunciado: Dado un conjunto de números, calcula el valor máximo.
Explicación: `max()` devuelve el elemento mayor del conjunto.

```
numeros = {5, 2, 9, 1}

mayor = max(numeros)

print(mayor)
```

Ejercicio 34

Enunciado: Dado un conjunto de números, calcula el valor mínimo.
Explicación: `min()` devuelve el elemento menor del conjunto.

```
numeros = {5, 2, 9, 1}
menor = min(numeros)
print(menor)
```

Ejercicio 35

Enunciado: Copia un conjunto y añade elementos a la copia sin afectar al original.
Explicación: `copy()` permite crear un nuevo conjunto independiente.

```
original = {1, 2, 3}
copia = original.copy()
copia.add(4)
print("Original:", original)
print("Copia:", copia)
```

Ejercicio 36

Enunciado: Convierte un conjunto en un string con elementos separados por comas.
Explicación: Primero se convierte cada elemento a cadena y luego se usa `join()`.

```
conjunto = {1, 2, 3}
resultado = ",".join(str(num) for num in conjunto)
print(resultado)
```

Ejercicio 37

Enunciado: Verifica si un conjunto está vacío.
Explicación: Un conjunto vacío tiene `len()` igual a 0.

```
conjunto = set()
print(len(conjunto) == 0)
```

Ejercicio 38

Enunciado: Combina tres conjuntos y elimina los duplicados automáticamente.
Explicación: `union()` puede recibir varios conjuntos y mantiene solo elementos únicos.

```
a = {1, 2}
b = {2, 3}
c = {3, 4}
resultado = a.union(b, c)
print(resultado)
```

Ejercicio 39

Enunciado: Dado un conjunto de palabras, muestra solo las palabras que contienen la letra “a”.
Explicación: Se recorre el conjunto y se filtra usando `if "a" in palabra`.

```
palabras = {"casa", "sol", "arena", "mar"}
con_a = {p for p in palabras if "a" in p}
print(con_a)
```

Ejercicio 40

Enunciado: Dado un conjunto de números, crea otro conjunto con los números al cuadrado.
Explicación: Se recorre el conjunto y se aplica `num**2` a cada elemento.

```
numeros = {1, 2, 3, 4}
cuadrados = {num**2 for num in numeros}
print(cuadrados)
```

Ejercicio 41

Enunciado: Dado un conjunto de números, muestra solo los números pares.
Explicación: Se recorre el conjunto y se filtran los elementos que sean divisibles entre 2.

```
numeros = {1, 2, 3, 4, 5, 6}
pares = {num for num in numeros if num % 2 == 0}
print(pares)
```

Ejercicio 42

Enunciado: Dado un conjunto de números, muestra solo los números impares.
Explicación: Se recorre el conjunto y se filtran los elementos que no sean divisibles entre 2.

```
numeros = {1, 2, 3, 4, 5, 6}
impares = {num for num in numeros if num % 2 != 0}
print(impares)
```

Ejercicio 43

Enunciado: Dado un conjunto de palabras, convierte todas a mayúsculas.
Explicación: Se recorre el conjunto y se aplica `upper()` a cada palabra.

```
palabras = {"hola", "mundo", "python"}
mayusculas = {p.upper() for p in palabras}
print(mayusculas)
```

Ejercicio 44

Enunciado: Dado un conjunto de palabras, convierte todas a minúsculas.
Explicación: Se recorre el conjunto y se aplica `lower()` a cada palabra.

```
palabras = {"HOLA", "MUNDO", "PYTHON"}
minusculas = {p.lower() for p in palabras}
print(minusculas)
```

Ejercicio 45

Enunciado: Dado un conjunto de números, elimina los números mayores que 5.
Explicación: Se recorre una copia del conjunto y se eliminan los elementos según la condición.

```
numeros = {1, 3, 6, 8, 4}
for num in numeros.copy():
    if num > 5:
        numeros.remove(num)
print(numeros)
```

Ejercicio 46

Enunciado: Dado un conjunto de números, añade todos los elementos de otra lista.
Explicación: `update()` agrega múltiples elementos de una lista al conjunto, sin duplicados.

```
conjunto = {1, 2}
lista = [2, 3, 4]
conjunto.update(lista)
print(conjunto)
```

Ejercicio 47

Enunciado: Dado un conjunto de números, muestra los elementos que no estén en otro conjunto.
Explicación: `difference()` devuelve los elementos que están en el primer conjunto pero no en el segundo.

```
a = {1, 2, 3, 4}
b = {3, 4, 5}
diff = a.difference(b)
print(diff)
```

Ejercicio 48

Enunciado: Dado un conjunto de números, muestra los elementos que están en ambos conjuntos.
Explicación: `intersection()` devuelve los elementos comunes a ambos conjuntos.

```
a = {1, 2, 3}
b = {2, 3, 4}
inter = a.intersection(b)
print(inter)
```

Ejercicio 49

Enunciado: Dado un conjunto de números, muestra los elementos que están en uno u otro conjunto, pero no en ambos.
Explicación: `symmetric_difference()` devuelve los elementos únicos de cada conjunto.

```
a = {1, 2, 3}
b = {3, 4, 5}
diff_sim = a.symmetric_difference(b)
print(diff_sim)
```

Ejercicio 50

Enunciado: Comprueba si dos conjuntos son iguales.
Explicación: El operador == compara si ambos conjuntos tienen exactamente los mismos elementos.

```
a = {1, 2, 3}
b = {3, 2, 1}
print(a == b)
```

Ejercicio 51

Enunciado: Dado un conjunto de números, elimina los elementos menores que 3.
Explicación: Se recorre una copia del conjunto y se eliminan los elementos según la condición.

```
numeros = {1, 2, 3, 4, 5}
for num in numeros.copy():
    if num < 3:
        numeros.remove(num)
print(numeros)
```

Ejercicio 52

Enunciado: Dado un conjunto de palabras, elimina las palabras que contengan la letra “a”.
Explicación: Se recorre una copia del conjunto y se eliminan los elementos que cumplan la condición.

```
palabras = {"casa", "sol", "perro", "mar"}
for p in palabras.copy():
    if "a" in p:
        palabras.remove(p)
print(palabras)
```

Ejercicio 53

Enunciado: Dado un conjunto de números, crea un conjunto con los números al cubo.
Explicación: Se usa comprensión de conjuntos para elevar cada elemento a la potencia 3.

```
numeros = {1, 2, 3, 4}
cubos = {num**3 for num in numeros}
print(cubos)
```

Ejercicio 54

Enunciado: Dado un conjunto de palabras, crea un conjunto con las palabras invertidas.
Explicación: Se usa comprensión de conjuntos y slicing `[::-1]` para invertir las palabras.

```
palabras = {"hola", "mundo", "python"}
invertidas = {p[::-1] for p in palabras}
print(invertidas)
```

Ejercicio 55

Enunciado: Dado un conjunto de números, comprueba si está vacío.
Explicación: Un conjunto vacío tiene `len()` igual a 0.

```
numeros = set()
print(len(numeros) == 0)
```

Ejercicio 56

Enunciado: Dado un conjunto de números, muestra el conjunto en orden ascendente como lista.
Explicación: `sorted()` devuelve una lista ordenada de los elementos del conjunto.

```
numeros = {5, 3, 1, 4, 2}
ordenados = sorted(numeros)
print(ordenados)
```

Ejercicio 57

Enunciado: Dado un conjunto de números, muestra el conjunto en orden descendente como lista.
Explicación: `sorted(..., reverse=True)` devuelve la lista de elementos en orden inverso.

```
numeros = {5, 3, 1, 4, 2}
descendentes = sorted(numeros, reverse=True)
print(descendentes)
```

Ejercicio 58

Enunciado: Dado un conjunto de palabras, muestra solo las palabras que tengan más de 4 letras.
Explicación: Se usa comprensión de conjuntos y `len()` para filtrar las palabras.

```
palabras = {"sol", "luna", "estrella", "mar"}
largas = {p for p in palabras if len(p) > 4}
print(largas)
```

Ejercicio 59

Enunciado: Dado un conjunto de números, crea un conjunto con los números multiplicados por 2.
Explicación: Se usa comprensión de conjuntos para duplicar cada elemento.

```
numeros = {1, 2, 3, 4}
dobles = {num*2 for num in numeros}
print(dobles)
```

Ejercicio 60

Enunciado: Dado un conjunto de palabras, crea un conjunto con las palabras en mayúsculas que tengan más de 3 letras.
Explicación: Se combina comprensión de conjuntos, `len()` y `upper()`.

```
palabras = {"sol", "luna", "estrella", "mar"}
resultado = {p.upper() for p in palabras if len(p) > 3}
print(resultado)
```

Ejercicio 61

Enunciado: Dado un conjunto de números, elimina todos los números pares.
Explicación: Se recorre una copia del conjunto y se elimina cada elemento que cumpla la condición `num % 2 == 0`.

```
numeros = {1, 2, 3, 4, 5, 6}
for num in numeros.copy():
    if num % 2 == 0:
        numeros.remove(num)
print(numeros)
```

Ejercicio 62

Enunciado: Dado un conjunto de números, elimina todos los números impares.
Explicación: Se recorre una copia del conjunto y se elimina cada elemento que cumpla la condición `num % 2 != 0`.

```
numeros = {1, 2, 3, 4, 5, 6}
for num in numeros.copy():
    if num % 2 != 0:
        numeros.remove(num)
print(numeros)
```

Ejercicio 63

Enunciado: Dado un conjunto de palabras, elimina las palabras que contengan más de 4 letras.
Explicación: Se recorre una copia del conjunto y se eliminan las palabras con `len(p) > 4`.

```
palabras = {"sol", "luna", "estrella", "mar"}
for p in palabras.copy():
    if len(p) > 4:
        palabras.remove(p)
print(palabras)
```

Ejercicio 64

Enunciado: Dado un conjunto de números, crea otro conjunto con los números negativos.
Explicación: Se usa comprensión de conjuntos para filtrar los números menores que 0.

```
numeros = {-3, -2, -1, 0, 1, 2, 3}
negativos = {num for num in numeros if num < 0}
print(negativos)
```

Ejercicio 65

Enunciado: Dado un conjunto de palabras, crea otro conjunto con las palabras que no contengan la letra “a”.
Explicación: Se filtran los elementos que no contienen la letra usando `if "a" not in p`.

```
palabras = {"casa", "sol", "perro", "mar"}
sin_a = {p for p in palabras if "a" not in p}
print(sin_a)
```

Ejercicio 66

Enunciado: Dado un conjunto de números, crea un conjunto con los números menores o iguales a 3.
Explicación: Se filtran los elementos usando la condición `num <= 3`.

```
numeros = {1, 2, 3, 4, 5}
menores_igual_3 = {num for num in numeros if num <= 3}
print(menores_igual_3)
```

Ejercicio 67

Enunciado: Dado un conjunto de palabras, elimina las palabras que empiecen con la letra “p”.
Explicación: Se recorre una copia del conjunto y se eliminan las palabras que cumplan la condición.

```
palabras = {"python", "programa", "hola", "mundo"}
for p in palabras.copy():
    if p.startswith("p"):
        palabras.remove(p)
print(palabras)
```

Ejercicio 68

Enunciado: Dado un conjunto de números, crea un conjunto con los números mayores que 5.
Explicación: Se usa comprensión de conjuntos para filtrar los elementos mayores que 5.

```
numeros = {2, 4, 6, 8, 10}
mayores_5 = {num for num in numeros if num > 5}
print(mayores_5)
```

Ejercicio 69

Enunciado: Dado un conjunto de palabras, crea un conjunto con las palabras que tengan exactamente 4 letras.
Explicación: Se filtran las palabras usando `len(p) == 4`.

```
palabras = {"sol", "luna", "estrella", "mar"}
exactamente_4 = {p for p in palabras if len(p) == 4}
print(exactamente_4)
```

Ejercicio 70

Enunciado: Dado un conjunto de números, crea un conjunto con los números multiplicados por 3.
Explicación: Se recorre el conjunto usando comprensión y se multiplica cada elemento por 3.

```
numeros = {1, 2, 3, 4}
triples = {num*3 for num in numeros}
print(triples)
```

Ejercicio 71

Enunciado: Dado un conjunto de números, muestra el conjunto en orden ascendente y descendente usando `sorted()`.

Explicación: `sorted()` devuelve una lista ordenada; `reverse=True` invierte el orden.

```
numeros = {5, 3, 8, 1, 2}
ascendente = sorted(numeros)
descendente = sorted(numeros, reverse=True)
print("Ascendente:", ascendente)
print("Descendente:", descendente)
```

Ejercicio 72

Enunciado: Dado un conjunto de palabras, crea un conjunto con las palabras en mayúsculas y minúsculas.

Explicación: Se usa comprensión de conjuntos combinando `upper()` y `lower()`.

```
palabras = {"Hola", "Mundo", "Python"}
resultado = {p.upper() for p in palabras} | {p.lower() for p in
palabras}
print(resultado)
```

Ejercicio 73

Enunciado: Dado un conjunto de números, crea otro conjunto con los números negativos y positivos por separado.

Explicación: Se usan dos comprensiones para filtrar negativos y positivos.

```
numeros = {-3, -2, -1, 0, 1, 2, 3}
negativos = {num for num in numeros if num < 0}
positivos = {num for num in numeros if num > 0}
print("Negativos:", negativos)
print("Positivos:", positivos)
```

Ejercicio 74

Enunciado: Dado un conjunto de números, elimina todos los múltiplos de 3.

Explicación: Se recorre una copia y se eliminan los elementos divisibles por 3.

```
numeros = {1, 2, 3, 4, 5, 6, 9}
for num in numeros.copy():
    if num % 3 == 0:
        numeros.remove(num)
print(numeros)
```

Ejercicio 75

Enunciado: Dado un conjunto de palabras, crea un conjunto con las palabras que contienen la letra “o”.
Explicación: Se filtran las palabras usando la condición `"o" in p`.

```
palabras = {"sol", "luna", "mar", "python"}
con_o = {p for p in palabras if "o" in p}
print(con_o)
```

Ejercicio 76

Enunciado: Dado un conjunto de palabras, elimina las palabras que contengan más de 5 letras.
Explicación: Se recorre una copia del conjunto y se eliminan las palabras con `len(p) > 5`.

```
palabras = {"sol", "luna", "estrella", "mar"}
for p in palabras.copy():
    if len(p) > 5:
        palabras.remove(p)
print(palabras)
```

Ejercicio 77

Enunciado: Dado un conjunto de números, crea un conjunto con los números al cuadrado si son pares.
Explicación: Se usa comprensión de conjuntos con la condición `num % 2 == 0`.

```
numeros = {1, 2, 3, 4, 5}
cuadrados_pares = {num**2 for num in numeros if num % 2 == 0}
print(cuadrados_pares)
```

Ejercicio 78

Enunciado: Dado un conjunto de números, crea un conjunto con los números al cuadrado si son impares.
Explicación: Se usa comprensión de conjuntos con la condición `num % 2 != 0`.

```
numeros = {1, 2, 3, 4, 5}
cuadrados_impares = {num**2 for num in numeros if num % 2 != 0}
print(cuadrados_impares)
```

Ejercicio 79

Enunciado: Dado un conjunto de números, muestra si un número específico está en el conjunto.
Explicación: Se usa el operador `in` para comprobar la pertenencia.

```
numeros = {1, 2, 3, 4, 5}
print(3 in numeros)
print(6 in numeros)
```

Ejercicio 80

Enunciado: Dado un conjunto de palabras, muestra cuántas palabras tienen más de 3 letras.
Explicación: Se usa comprensión de conjuntos y `len()` para contar los elementos que cumplen la condición.

```
palabras = {"sol", "luna", "estrella", "mar"}
resultado = len({p for p in palabras if len(p) > 3})
print(resultado)
```

Ejercicio 81

Enunciado: Dado un conjunto de números, crea un conjunto con los números mayores que 3 y menores que 7.
Explicación: Se usa comprensión de conjuntos con una condición combinada `num > 3 and num < 7`.

```
numeros = {1, 2, 3, 4, 5, 6, 7, 8}
resultado = {num for num in numeros if num > 3 and num < 7}
print(resultado)
```

Ejercicio 82

Enunciado: Dado un conjunto de palabras, crea un conjunto con las palabras que empiecen con “m” o “p”.
Explicación: Se filtran los elementos usando `startswith()`.

```
palabras = {"python", "mundo", "casa", "programa"}
resultado = {p for p in palabras if p.startswith("m") or
p.startswith("p")}
print(resultado)
```

Ejercicio 83

Enunciado: Dado un conjunto de números, crea un conjunto con los números que sean múltiplos de 2 o 3.
Explicación: Se filtran los elementos usando la condición `num % 2 == 0 or num % 3 == 0`.

```
numeros = {1, 2, 3, 4, 5, 6, 7}
resultado = {num for num in numeros if num % 2 == 0 or num % 3 == 0}
print(resultado)
```

Ejercicio 84

Enunciado: Dado un conjunto de palabras, convierte todas las palabras a mayúsculas y elimina las que tengan más de 4 letras.
Explicación: Se combina comprensión de conjuntos con `upper()` y `len()`.

```
palabras = {"sol", "luna", "estrella", "mar"}
resultado = {p.upper() for p in palabras if len(p) <= 4}
print(resultado)
```

Ejercicio 85

Enunciado: Dado un conjunto de números, crea un conjunto con los números que no sean múltiplos de 5.
Explicación: Se filtran los elementos usando la condición `num % 5 != 0`.

```
numeros = {5, 10, 12, 15, 20, 23}
resultado = {num for num in numeros if num % 5 != 0}
print(resultado)
```

Ejercicio 86

Enunciado: Dado un conjunto de palabras, crea un conjunto con las palabras que tengan la letra “o” pero no la letra “a”.
Explicación: Se filtran los elementos usando la condición `"o" in p and "a" not in p`.

```
palabras = {"sol", "casa", "mundo", "mar"}
resultado = {p for p in palabras if "o" in p and "a" not in p}
print(resultado)
```

Ejercicio 87

Enunciado: Dado un conjunto de números, crea un conjunto con los números pares al cuadrado.
Explicación: Se usa comprensión de conjuntos con condición `num % 2 == 0` y `num**2`.

```
numeros = {1, 2, 3, 4, 5, 6}
resultado = {num**2 for num in numeros if num % 2 == 0}
print(resultado)
```

Ejercicio 88

Enunciado: Dado un conjunto de palabras, elimina las palabras que terminen en “a”.
Explicación: Se recorre una copia del conjunto y se eliminan los elementos que cumplan la condición `endswith("a")`.

```
palabras = {"casa", "sol", "luna", "mar"}
for p in palabras.copy():
    if p.endswith("a"):
        palabras.remove(p)
print(palabras)
```

Ejercicio 89

Enunciado: Dado un conjunto de números, crea un conjunto con los números que sean primos.
Explicación: Se filtran los elementos usando una función simple que verifica si un número es primo.

```
def es_primo(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5)+1):
        if n % i == 0:
            return False
    return True

numeros = {1, 2, 3, 4, 5, 6, 7}
primos = {num for num in numeros if es_primo(num)}
print(primos)
```

Ejercicio 90

Enunciado: Dado un conjunto de palabras, crea un conjunto con las palabras que contengan al menos una vocal.
Explicación: Se filtran los elementos usando `any()` y una lista de vocales.

```
palabras = {"sol", "mnr", "luz", "python"}
vocales = "aeiou"
resultado = {p for p in palabras if any(v in p for v in vocales)}
print(resultado)
```

Ejercicio 91

Enunciado: Dado un conjunto de números, crea un conjunto con los números menores que 5.
Explicación: Se filtran los elementos usando la condición `num < 5`.

```
numeros = {1, 3, 5, 7, 9}
resultado = {num for num in numeros if num < 5}
print(resultado)
```

Ejercicio 92

Enunciado: Dado un conjunto de palabras, crea un conjunto con las palabras que tengan exactamente 5 letras.
Explicación: Se filtran los elementos usando `len(p) == 5`.

```
palabras = {"sol", "luna", "estrella", "perro", "gato"}
resultado = {p for p in palabras if len(p) == 5}
print(resultado)
```

Ejercicio 93

Enunciado: Dado un conjunto de números, crea un conjunto con los números divisibles por 2 y 3 a la vez.
Explicación: Se filtran los elementos usando la condición `num % 2 == 0 and num % 3 == 0`.

```
numeros = {1, 2, 3, 4, 5, 6, 12, 15}
resultado = {num for num in numeros if num % 2 == 0 and num % 3 == 0}
print(resultado)
```

Ejercicio 94

Enunciado: Dado un conjunto de palabras, crea un conjunto con las palabras que tengan vocales “a” o “e”.
Explicación: Se filtran los elementos usando `any(v in p for v in "ae")`.

```
palabras = {"sol", "casa", "luz", "perro"}
resultado = {p for p in palabras if any(v in p for v in "ae")}
print(resultado)
```

Ejercicio 95

Enunciado: Dado un conjunto de números, crea un conjunto con los números multiplicados por 5.
Explicación: Se recorre el conjunto usando comprensión y se multiplica cada elemento por 5.

```
numeros = {1, 2, 3, 4}
resultado = {num*5 for num in numeros}
print(resultado)
```

Ejercicio 96

Enunciado: Dado un conjunto de palabras, crea un conjunto con las palabras invertidas que tengan más de 3 letras.
Explicación: Se usa comprensión de conjuntos, `len()` y slicing `[::-1]`.

```
palabras = {"sol", "luna", "estrella", "mar"}
resultado = {p[::-1] for p in palabras if len(p) > 3}
print(resultado)
```

Ejercicio 97

Enunciado: Dado un conjunto de números, elimina los números que estén en otro conjunto.
Explicación: Se usa `difference_update()` para eliminar elementos de un conjunto que están en otro.

```
a = {1, 2, 3, 4, 5}
b = {2, 4}
a.difference_update(b)
print(a)
```

Ejercicio 98

Enunciado: Dado un conjunto de palabras, elimina las palabras que estén en otro conjunto.
Explicación: Se usa `difference_update()` para eliminar elementos de un conjunto que están en otro.

```
a = {"sol", "luna", "mar"}
b = {"luna", "mar"}
a.difference_update(b)
print(a)
```

Ejercicio 99

Enunciado: Dado un conjunto de números, crea un conjunto con los números que no estén en otro conjunto.
Explicación: `difference()` devuelve un nuevo conjunto con los elementos del primer conjunto que no están en el segundo.

```
a = {1, 2, 3, 4, 5}
b = {3, 4}
resultado = a.difference(b)
print(resultado)
```

Ejercicio 100

Enunciado: Dado un conjunto de palabras, une dos conjuntos y elimina duplicados automáticamente.
Explicación: `union()` combina todos los elementos de ambos conjuntos y elimina duplicados.

```
a = {"sol", "luna"}
b = {"luna", "mar", "estrella"}
resultado = a.union(b)
print(resultado)
```

Ejercicios python tuplas

Ejercicio 1

Enunciado: Crea una tupla con cinco números e imprímela.
Explicación: Una tupla se define usando paréntesis `()` y puede contener varios elementos.

```
numeros = (1, 2, 3, 4, 5)
print(numeros)
```

Ejercicio 2

Enunciado: Accede al primer y último elemento de una tupla.
Explicación: Se usan índices como en las listas, empezando en 0 y con `-1` para el último elemento.

```
tupla = ("sol", "luna", "estrella", "mar")
print("Primero:", tupla[0])
print("Último:", tupla[-1])
```

Ejercicio 3

Enunciado: Cuenta cuántas veces aparece un elemento en una tupla.
Explicación: Se usa el método `.count()` que devuelve el número de apariciones.

```
tupla = (1, 2, 2, 3, 4, 2)
print("El número 2 aparece:", tupla.count(2), "veces")
```

Ejercicio 4

Enunciado: Muestra la longitud de una tupla.
Explicación: Se usa la función `len()` para saber cuántos elementos tiene la tupla.

```
tupla = ("python", "java", "c++", "go")
print("Número de elementos:", len(tupla))
```

Ejercicio 5

Enunciado: Comprueba si un elemento está en una tupla.
Explicación: Se usa el operador `in` para verificar si un elemento existe dentro de la tupla.

```
tupla = ("manzana", "pera", "uva")
print("manzana" in tupla)
print("melon" in tupla)
```

Ejercicio 6

Enunciado: Concatena dos tuplas.
Explicación: Se puede usar el operador `+` para unir dos tuplas en una sola.

```
tupla1 = (1, 2, 3)
tupla2 = (4, 5, 6)
resultado = tupla1 + tupla2
print(resultado)
```

Ejercicio 7

Enunciado: Multiplica una tupla por un número.
Explicación: Multiplicar una tupla repite sus elementos la cantidad indicada.

```
tupla = ("A", "B")
resultado = tupla * 3
print(resultado)
```

Ejercicio 8

Enunciado: Convierte una lista en tupla.
Explicación: Se usa la función `tuple()` para transformar una lista en tupla.

```
lista = [1, 2, 3, 4]
tupla = tuple(lista)
print(tupla)
```

Ejercicio 9

Enunciado: Convierte una tupla en lista.
Explicación: Se usa la función `list()` para poder modificar los elementos si es necesario.

```
tupla = (10, 20, 30)
lista = list(tupla)
print(lista)
```

Ejercicio 10

Enunciado: Crea una tupla con diferentes tipos de datos y muéstrala.
Explicación: Las tuplas pueden contener elementos de distintos tipos (int, str, bool, etc.).

```
tupla = (1, "hola", True, 3.14)
print(tupla)
```

Ejercicio 11

Enunciado: Muestra los elementos de una tupla usando un bucle `for`.
Explicación: Se puede recorrer una tupla elemento por elemento con un bucle `for`.

```
tupla = ("rojo", "verde", "azul")
for color in tupla:
    print(color)
```

Ejercicio 12

Enunciado: Muestra los elementos de una tupla junto con sus posiciones.
Explicación: Se usa la función `enumerate()` para obtener índice y valor.

```
tupla = ("sol", "luna", "mar")
for i, valor in enumerate(tupla):
    print("Índice:", i, "Valor:", valor)
```

Ejercicio 13

Enunciado: Crea una tupla a partir de un rango de números.
Explicación: Se puede usar la función `tuple()` junto a `range()` para generar una secuencia inmutable.

```
tupla = tuple(range(1, 6))
print(tupla)
```

Ejercicio 14

Enunciado: Obtén una porción (slice) de una tupla.
Explicación: Usamos la notación `[inicio:fin]` para tomar una parte de la tupla.

```
tupla = (10, 20, 30, 40, 50)
sub_tupla = tupla[1:4]
print(sub_tupla)
```

Ejercicio 15

Enunciado: Muestra los elementos de una tupla en orden inverso.
Explicación: Con slicing `[::-1]` invertimos el orden de los elementos.

```
tupla = (1, 2, 3, 4, 5)
inversa = tupla[::-1]
print(inversa)
```

Ejercicio 16

Enunciado: Desempaqueta una tupla en variables individuales.
Explicación: Se puede asignar cada elemento de una tupla a una variable.

```
tupla = ("python", 3.11, "lenguaje")
nombre, version, tipo = tupla
print(nombre, version, tipo)
```

Ejercicio 17

Enunciado: Encuentra el valor máximo y mínimo en una tupla de números.
Explicación: Se usan las funciones `max()` y `min()`.

```
tupla = (10, 5, 8, 12, 3)
print("Máximo:", max(tupla))
print("Mínimo:", min(tupla))
```

Ejercicio 18

Enunciado: Suma todos los valores de una tupla de números.
Explicación: La función `sum()` devuelve la suma total de los elementos.

```
tupla = (4, 7, 9, 3)
print("Suma total:", sum(tupla))
```

Ejercicio 19

Enunciado: Crea una tupla con números del 1 al 10 y muestra solo los pares.
Explicación: Se usa comprensión de tuplas mediante `tuple()` y una lista por comprensión.

```
tupla = tuple(range(1, 11))
pares = tuple(num for num in tupla if num % 2 == 0)
print(pares)
```

Ejercicio 20

Enunciado: Comprueba si una tupla está vacía o no.
Explicación: Una tupla vacía se define como `()` y se evalúa con un condicional `if`.

```
tupla = ()
if not tupla:
    print("La tupla está vacía")
else:
    print("La tupla tiene elementos")
```

Ejercicio 21

Enunciado: Muestra solo los elementos únicos de una tupla con valores repetidos.
Explicación: Se puede convertir la tupla en conjunto `set()` para eliminar duplicados.

```
tupla = (1, 2, 2, 3, 4, 4, 5)
unicos = tuple(set(tupla))
print(unicos)
```

Ejercicio 22

Enunciado: Combina dos tuplas de igual longitud en una sola de pares.
Explicación: Se usa la función `zip()` para agrupar los elementos por posición.

```
nombres = ("Ana", "Luis", "Marta")
edades = (20, 25, 30)
combinadas = tuple(zip(nombres, edades))
print(combinadas)
```

Ejercicio 23

Enunciado: Crea una tupla con los cuadrados de los números del 1 al 5.
Explicación: Se usa comprensión de tuplas con `tuple()` y una lista por comprensión.

```
tupla = tuple(x**2 for x in range(1, 6))
print(tupla)
```

Ejercicio 24

Enunciado: Cuenta cuántas veces aparece un nombre en una tupla de nombres.
Explicación: Se usa el método `.count()` para obtener el número de apariciones.

```
tupla = ("Ana", "Luis", "Ana", "Marta", "Ana")
print("Ana aparece:", tupla.count("Ana"), "veces")
```

Ejercicio 25

Enunciado: Encuentra el índice de un elemento dentro de una tupla.
Explicación: Se usa el método `.index()` que devuelve la posición del primer elemento encontrado.

```
tupla = (10, 20, 30, 40)
print("El 30 está en la posición:", tupla.index(30))
```

Ejercicio 26

Enunciado: Crea una tupla con los caracteres de una palabra.
Explicación: La función `tuple()` convierte la cadena en una tupla de caracteres.

```
palabra = "python"
tupla = tuple(palabra)
print(tupla)
```

Ejercicio 27

Enunciado: Crea una tupla que contenga el doble de cada número de otra tupla.
Explicación: Se usa comprensión de tuplas para multiplicar cada elemento.

```
tupla = (1, 2, 3, 4)
dobles = tuple(x*2 for x in tupla)
print(dobles)
```

Ejercicio 28

Enunciado: Suma elemento a elemento dos tuplas del mismo tamaño.
Explicación: Se usa `zip()` y comprensión de tuplas para sumar por pares.

```
a = (1, 2, 3)
b = (4, 5, 6)
suma = tuple(x + y for x, y in zip(a, b))
print(suma)
```

Ejercicio 29

Enunciado: Verifica si todos los elementos de una tupla son números positivos.
Explicación: Se usa `all()` con una condición dentro de una comprensión.

```
tupla = (2, 4, 6, 8)
if all(x > 0 for x in tupla):
    print("Todos los números son positivos")
else:
    print("Hay algún número negativo o cero")
```

Ejercicio 30

Enunciado: Verifica si al menos un número de una tupla es impar.
Explicación: Se usa `any()` para comprobar si alguna condición se cumple.

```
tupla = (2, 4, 6, 9)
if any(x % 2 != 0 for x in tupla):
    print("Hay al menos un número impar")
else:
    print("Todos son pares")
```

Ejercicio 31

Enunciado: Crea una tupla con los días de la semana y muestra el tercer día.
Explicación: Se accede a la posición usando el índice correspondiente.

```
dias = ("Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado",
"Domingo")
print("El tercer día es:", dias[2])
```

Ejercicio 32

Enunciado: Muestra la cantidad de elementos que tiene una tupla de meses.
Explicación: Se usa la función `len()` para conocer su longitud.

```
meses = ("Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio")
print("La tupla tiene", len(meses), "meses")
```

Ejercicio 33

Enunciado: Crea una tupla con las vocales y comprueba si la letra “e” está en ella.
Explicación: Se utiliza el operador `in` para verificar la existencia de un elemento.

```
vocales = ("a", "e", "i", "o", "u")
print("¿Está la 'e'?:", "e" in vocales)
```

Ejercicio 34

Enunciado: Crea una tupla de números y muestra los números mayores que 5.
Explicación: Se usa una comprensión de tuplas con una condición.

```
tupla = (1, 3, 5, 7, 9)

mayores = tuple(x for x in tupla if x > 5)

print(mayores)
```

Ejercicio 35

Enunciado: Convierte una cadena con números separados por comas en una tupla de enteros.
Explicación: Se usa `split()` y `map()` junto a `tuple()`.

```
cadena = "1,2,3,4,5"

tupla = tuple(map(int, cadena.split(",")))

print(tupla)
```

Ejercicio 36

Enunciado: Crea una tupla con palabras y muestra la más larga.
Explicación: Se usa la función `max()` con el argumento `key=len`.

```
palabras = ("sol", "montaña", "mar", "estrella")

larga = max(palabras, key=len)

print("La palabra más larga es:", larga)
```

Ejercicio 37

Enunciado: Crea una tupla con números del 1 al 10 y muestra el promedio.
Explicación: Se usa `sum()` y `len()` para calcular la media aritmética.

```
tupla = tuple(range(1, 11))

promedio = sum(tupla) / len(tupla)

print("El promedio es:", promedio)
```

Ejercicio 38

Enunciado: Crea una tupla con varias cadenas y muéstralas en mayúsculas.
Explicación: Se usa comprensión de tuplas y el método `.upper()`.

```
tupla = ("hola", "python", "tuplas")

mayusculas = tuple(p.upper() for p in tupla)

print(mayusculas)
```

Ejercicio 39

Enunciado: Crea una tupla con números y muestra solo los múltiplos de 3.
Explicación: Se usa comprensión de tuplas con una condición `num % 3 == 0`.

```
tupla = tuple(range(1, 16))
multiplos = tuple(num for num in tupla if num % 3 == 0)
print(multiplos)
```

Ejercicio 40

Enunciado: Une tres tuplas en una sola.
Explicación: Se utiliza el operador `+` para concatenar múltiples tuplas.

```
a = (1, 2)
b = (3, 4)
c = (5, 6)
resultado = a + b + c
print(resultado)
```

Ejercicio 41

Enunciado: Crea una tupla con nombres y muestra cuántos nombres tienen más de 4 letras.
Explicación: Se usa una comprensión de tuplas y la función `len()` para contar.

```
nombres = ("Ana", "Luis", "Marta", "Pedro", "Eva")
mayores = tuple(nombre for nombre in nombres if len(nombre) > 4)
print("Nombres con más de 4 letras:", mayores)
```

Ejercicio 42

Enunciado: Crea una tupla con números y muestra la suma de los pares.
Explicación: Se usa una comprensión y `sum()` para obtener la suma de los números pares.

```
tupla = (1, 2, 3, 4, 5, 6)
suma_pares = sum(x for x in tupla if x % 2 == 0)
print("Suma de los números pares:", suma_pares)
```

Ejercicio 43

Enunciado: Crea una tupla con tres coordenadas y desempaquétalas en variables.
Explicación: El desempaquetado permite asignar los valores directamente.

```
coordenadas = (10, 20, 30)
x, y, z = coordenadas
print("x =", x, "y =", y, "z =", z)
```

Ejercicio 44

Enunciado: Crea una tupla con números y encuentra su mediana.
Explicación: Se ordena la tupla y se obtiene el valor central.

```
tupla = (5, 1, 9, 3, 7)
ordenada = sorted(tupla)
mediana = ordenada[len(ordenada)//2]
print("Mediana:", mediana)
```

Ejercicio 45

Enunciado: Crea una tupla con cadenas y muestra cuántas comienzan con vocal.
Explicación: Se usa comprensión y el método `.startswith()` para comprobar.

```
palabras = ("sol", "agua", "estrella", "cielo", "oso")
vocales = tuple(p for p in palabras if p[0].lower() in "aeiou")
print("Palabras que empiezan con vocal:", vocales)
```

Ejercicio 46

Enunciado: Crea una tupla con números aleatorios y muestra el mayor.
Explicación: Se genera con `random.randint()` y se usa `max()`.

```
import random
tupla = tuple(random.randint(1, 50) for _ in range(5))
print("Tupla:", tupla)
print("Número mayor:", max(tupla))
```

Ejercicio 47

Enunciado: Crea una tupla con cadenas y ordénala alfabéticamente.
Explicación: Se usa `sorted()` para devolver una lista ordenada y `tuple()` para convertirla.

```
tupla = ("pera", "manzana", "uva", "banana")
ordenada = tuple(sorted(tupla))
print(ordenada)
```

Ejercicio 48

Enunciado: Crea una tupla con números y elimina duplicados convirtiéndola a conjunto.
Explicación: Los conjuntos no permiten elementos repetidos.

```
tupla = (1, 2, 2, 3, 3, 4)
sin_duplicados = tuple(set(tupla))
print(sin_duplicados)
```

Ejercicio 49

Enunciado: Crea una tupla de nombres y muéstrala en orden inverso.
Explicación: Se usa slicing `[::-1]` para invertir los elementos.

```
nombres = ("Ana", "Luis", "Marta", "Carlos")
invertida = nombres[::-1]
print(invertida)
```

Ejercicio 50

Enunciado: Crea una tupla con cinco números y muestra el segundo mayor.
Explicación: Se ordena la tupla y se accede al penúltimo elemento.

```
tupla = (8, 3, 5, 1, 9)
ordenada = sorted(tupla)
print("Segundo mayor:", ordenada[-2])
```

Ejercicio 51

Enunciado: Crea una tupla con temperaturas y muestra la más baja.
Explicación: Se usa la función `min()` para encontrar el menor valor.

```
temperaturas = (22, 18, 25, 19, 30)
print("Temperatura más baja:", min(temperaturas))
```

Ejercicio 52

Enunciado: Crea una tupla con varios números y calcula el producto de todos ellos.
Explicación: Se usa un bucle `for` para multiplicar cada elemento.

```
tupla = (2, 3, 4)
producto = 1
for n in tupla:
    producto *= n
print("Producto:", producto)
```

Ejercicio 53

Enunciado: Crea una tupla con nombres y muestra el primero en orden alfabético.
Explicación: Se usa la función `min()` porque las cadenas se ordenan alfabéticamente.

```
nombres = ("Pedro", "Ana", "Luis", "Marta")
print("Primer nombre en orden alfabético:", min(nombres))
```

Ejercicio 54

Enunciado: Crea una tupla con palabras y muestra cuántas tienen más de 3 letras.
Explicación: Se usa comprensión y la función `len()`.

```
palabras = ("sol", "mar", "estrella", "luz", "luna")
largas = tuple(p for p in palabras if len(p) > 3)
print("Palabras con más de 3 letras:", largas)
```

Ejercicio 55

Enunciado: Crea una tupla con edades y muestra cuántas son mayores de edad (≥18).
Explicación: Se filtran los valores que cumplen la condición.

```
edades = (12, 18, 25, 30, 15)
mayores = tuple(e for e in edades if e >= 18)
print("Mayores de edad:", mayores)
```

Ejercicio 56

Enunciado: Crea una tupla con los números del 1 al 10 y muestra los impares.
Explicación: Se usa comprensión de tuplas con la condición `x % 2 != 0`.

```
tupla = tuple(range(1, 11))
impares = tuple(x for x in tupla if x % 2 != 0)
print("Números impares:", impares)
```

Ejercicio 57

Enunciado: Crea una tupla con números y muestra el valor absoluto de cada uno.
Explicación: Se usa la función `abs()` dentro de una comprensión de tuplas.

```
tupla = (-3, -1, 0, 2, 5)
absolutos = tuple(abs(x) for x in tupla)
print("Valores absolutos:", absolutos)
```

Ejercicio 58

Enunciado: Crea una tupla con nombres y muestra el más largo.
Explicación: Se usa `max()` con el parámetro `key=len`.

```
nombres = ("Ana", "Roberto", "Luis", "Marta")
largo = max(nombres, key=len)
print("Nombre más largo:", largo)
```

Ejercicio 59

Enunciado: Crea una tupla con cadenas y calcula el total de caracteres.
Explicación: Se suman las longitudes de todas las cadenas con `sum()` y `len()`.

```
palabras = ("sol", "luna", "estrella")
total = sum(len(p) for p in palabras)
print("Total de caracteres:", total)
```

Ejercicio 60

Enunciado: Crea una tupla con 5 números y muestra la suma, el promedio y el máximo.
Explicación: Se usan las funciones `sum()`, `len()` y `max()`.

```
tupla = (4, 8, 6, 2, 10)
print("Suma:", sum(tupla))
print("Promedio:", sum(tupla)/len(tupla))
print("Máximo:", max(tupla))
```

Ejercicio 61

Enunciado: Crea una tupla con 5 colores e imprime el segundo y cuarto color.
Explicación: Se accede por índice para mostrar los elementos deseados.

```
colores = ("rojo", "verde", "azul", "amarillo", "negro")
print("Segundo color:", colores[1])
print("Cuarto color:", colores[3])
```

Ejercicio 62

Enunciado: Crea una tupla con números del 1 al 10 y muestra el último elemento.
Explicación: Se usa el índice negativo `-1` para acceder al último valor.

```
tupla = tuple(range(1, 11))
print("Último elemento:", tupla[-1])
```

Ejercicio 63

Enunciado: Crea una tupla con letras y muestra la cantidad de veces que aparece una letra dada.
Explicación: Se usa el método `.count()` para contar apariciones.

```
letras = ("a", "b", "a", "c", "a", "d")
print("La letra 'a' aparece:", letras.count("a"), "veces")
```

Ejercicio 64

Enunciado: Crea una tupla con números y muestra el índice del número 5.
Explicación: Se usa el método `.index()` para obtener la posición del elemento.

```
tupla = (3, 5, 7, 5, 9)

print("El número 5 aparece por primera vez en la posición:", tupla.index(5))
```

Ejercicio 65

Enunciado: Crea una tupla con números y ordénala de menor a mayor.
Explicación: Se usa la función `sorted()` y luego se convierte en tupla.

```
tupla = (9, 3, 1, 7, 5)

ordenada = tuple(sorted(tupla))

print("Tupla ordenada:", ordenada)
```

Ejercicio 66

Enunciado: Crea una tupla con números y muestra cuántos son mayores que 50.
Explicación: Se usa comprensión y una condición dentro de `tuple()`.

```
tupla = (10, 60, 75, 30, 90)

mayores = tuple(x for x in tupla if x > 50)

print("Números mayores que 50:", mayores)
```

Ejercicio 67

Enunciado: Crea una tupla con números negativos y conviértelos a positivos.
Explicación: Se usa `abs()` para convertir cada número a su valor absoluto.

```
tupla = (-4, -2, 0, 3, -1)

positivos = tuple(abs(x) for x in tupla)

print("Números positivos:", positivos)
```

Ejercicio 68

Enunciado: Crea una tupla con nombres y ordénalos por longitud.
Explicación: Se usa `sorted()` con el argumento `key=len`.

```
nombres = ("Ana", "Cristina", "Luis", "Bea")
ordenados = tuple(sorted(nombres, key=len))
print("Nombres ordenados por longitud:", ordenados)
```

Ejercicio 69

Enunciado: Crea una tupla con edades y calcula cuántas son menores de 18.
Explicación: Se usa comprensión y `len()` para contar los valores que cumplen la condición.

```
edades = (12, 25, 17, 30, 15, 19)
menores = tuple(e for e in edades if e < 18)
print("Edades menores de 18:", menores)
```

Ejercicio 70

Enunciado: Crea una tupla con números y muestra el promedio de los pares.
Explicación: Se filtran los pares y se calcula la media.

```
tupla = (2, 3, 4, 5, 6, 7)
pares = [x for x in tupla if x % 2 == 0]
promedio = sum(pares) / len(pares)
print("Promedio de números pares:", promedio)
```

Ejercicio 71

Enunciado: Crea una tupla con los días de la semana e imprime los tres primeros.
Explicación: Se utiliza el operador de corte `[:3]` para mostrar una parte de la tupla.

```
dias = ("lunes", "martes", "miércoles", "jueves", "viernes",
"sábado", "domingo")
print("Tres primeros días:", dias[:3])
```

Ejercicio 72

Enunciado: Crea una tupla con frutas y muestra si “manzana” está en la tupla.
Explicación: Se usa el operador `in` para comprobar la pertenencia.

```
frutas = ("pera", "manzana", "uva", "naranja")
if "manzana" in frutas:
    print("Manzana está en la tupla")
else:
    print("Manzana no está en la tupla")
```

Ejercicio 73

Enunciado: Crea dos tuplas con números y forma una tercera concatenando ambas.
Explicación: Se usa el operador `+` para unir tuplas.

```
a = (1, 2, 3)
b = (4, 5, 6)
c = a + b
print("Tupla combinada:", c)
```

Ejercicio 74

Enunciado: Crea una tupla con palabras e imprime las que tengan más de 4 letras.
Explicación: Se usa una comprensión con `len()` para filtrar palabras.

```
palabras = ("sol", "nube", "montaña", "mar", "río")
largas = tuple(p for p in palabras if len(p) > 4)
print("Palabras con más de 4 letras:", largas)
```

Ejercicio 75

Enunciado: Crea una tupla con números y encuentra el máximo y mínimo.
Explicación: Se usan las funciones `max()` y `min()` para obtener los extremos.

```
numeros = (10, 25, 3, 48, 17)
print("Número máximo:", max(numeros))
print("Número mínimo:", min(numeros))
```

Ejercicio 76

Enunciado: Crea una tupla con 6 números y muestra la suma de los tres primeros.
Explicación: Se aplica `sum()` sobre un corte de la tupla.

```
tupla = (5, 10, 15, 20, 25, 30)
print("Suma de los tres primeros:", sum(tupla[:3]))
```

Ejercicio 77

Enunciado: Crea una tupla con 6 números y muestra la suma de los tres últimos.
Explicación: Se usa el operador de corte `[-3:]`.

```
tupla = (5, 10, 15, 20, 25, 30)
print("Suma de los tres últimos:", sum(tupla[-3:]))
```

Ejercicio 78

Enunciado: Crea una tupla con nombres y muestra el más largo.
Explicación: Se usa `max()` con el argumento `key=len`.

```
nombres = ("Ana", "Roberto", "María", "José")
print("Nombre más largo:", max(nombres, key=len))
```

Ejercicio 79

Enunciado: Crea una tupla con precios y muestra los que están entre 10 y 50.
Explicación: Se usa una comprensión con condiciones compuestas.

```
precios = (5, 10, 25, 50, 60, 45)
rango = tuple(p for p in precios if 10 <= p <= 50)
print("Precios entre 10 y 50:", rango)
```

Ejercicio 80

Enunciado: Crea una tupla con números del 1 al 10 e imprime los impares.
Explicación: Se filtran con una condición que verifica `x % 2 != 0`.

```
numeros = tuple(range(1, 11))
impares = tuple(x for x in numeros if x % 2 != 0)
print("Números impares:", impares)
```

Ejercicio 81

Enunciado: Crea una tupla con temperaturas y muestra la media.
Explicación: Se usa `sum()` y `len()` para calcular el promedio.

```
temperaturas = (20, 25, 22, 18, 24)
media = sum(temperaturas) / len(temperaturas)
print("Temperatura media:", media)
```

Ejercicio 82

Enunciado: Crea una tupla con números y determina si todos son positivos.
Explicación: Se usa `all()` con una condición booleana.

```
numeros = (3, 7, 2, 9, 4)
if all(x > 0 for x in numeros):
    print("Todos los números son positivos")
else:
    print("Hay números negativos o cero")
```

Ejercicio 83

Enunciado: Crea una tupla con nombres e imprime solo los que comiencen con “M”.
Explicación: Se usa una comprensión filtrando por la primera letra.

```
nombres = ("María", "Luis", "Marta", "Pedro", "Miguel")
con_m = tuple(n for n in nombres if n.startswith("M"))
print("Nombres que empiezan con M:", con_m)
```

Ejercicio 84

Enunciado: Crea una tupla con números y muestra el producto de todos ellos.
Explicación: Se recorre la tupla multiplicando los elementos.

```
numeros = (2, 3, 4, 5)
producto = 1
for n in numeros:
    producto *= n
print("Producto total:", producto)
```

Ejercicio 85

Enunciado: Crea una tupla con nombres de ciudades y muestra la cantidad total de letras.
Explicación: Se suman las longitudes de cada nombre con `sum()` y `len()`.

```
ciudades = ("Zaragoza", "Madrid", "Sevilla", "Bilbao")
total_letras = sum(len(c) for c in ciudades)
print("Cantidad total de letras:", total_letras)
```

Ejercicio 86

Enunciado: Crea una tupla con números y genera otra con sus cuadrados.
Explicación: Se usa comprensión de tupla aplicando `x**2`.

```
numeros = (1, 2, 3, 4, 5)
cuadrados = tuple(x**2 for x in numeros)
print("Cuadrados:", cuadrados)
```

Ejercicio 87

Enunciado: Crea una tupla con números y elimina los duplicados convirtiéndola en conjunto.
Explicación: Se convierte a `set` y luego de nuevo a `tuple`.

```
tupla = (1, 2, 2, 3, 4, 4, 5)
sin_duplicados = tuple(set(tupla))
print("Sin duplicados:", sin_duplicados)
```

Ejercicio 88

Enunciado: Crea una tupla con 5 números y muestra si contiene el número 10.
Explicación: Se usa el operador `in` para verificar la existencia.

```
numeros = (4, 8, 15, 16, 23)
print("¿Contiene el número 10?", 10 in numeros)
```

Ejercicio 89

Enunciado: Crea una tupla con edades y muestra las mayores de 30 en orden ascendente.
Explicación: Se filtran con una comprensión y luego se ordenan con `sorted()`.

```
edades = (25, 40, 19, 35, 50, 28)
mayores = tuple(sorted(e for e in edades if e > 30))
print("Edades mayores de 30:", mayores)
```

Ejercicio 90

Enunciado: Crea una tupla con nombres y muestra los que terminen en “a”.
Explicación: Se usa `endswith()` para filtrar por la última letra.

```
nombres = ("Lucía", "Carlos", "Marta", "Elena", "José")
terminan_a = tuple(n for n in nombres if n.endswith("a"))
print("Nombres que terminan en 'a':", terminan_a)
```

Ejercicio 91

Enunciado: Crea una tupla con 6 números y muestra solo los pares.
Explicación: Se filtran con una comprensión verificando `x % 2 == 0`.

```
numeros = (1, 2, 3, 4, 5, 6)
pares = tuple(x for x in numeros if x % 2 == 0)
print("Números pares:", pares)
```

Ejercicio 92

Enunciado: Crea una tupla con frutas y muestra la cantidad total de elementos.
Explicación: Se usa la función `len()` para contar los elementos.

```
frutas = ("pera", "manzana", "uva", "naranja", "plátano")
print("Cantidad de frutas:", len(frutas))
```

Ejercicio 93

Enunciado: Crea una tupla con números y muestra si contiene valores repetidos.
Explicación: Se compara la longitud de la tupla con la de su conversión a `set()`.

```
tupla = (1, 2, 3, 2, 4)
if len(tupla) != len(set(tupla)):
    print("Hay valores repetidos")
else:
    print("No hay valores repetidos")
```

Ejercicio 94

Enunciado: Crea una tupla con números y muestra el segundo mayor.
Explicación: Se convierte en lista ordenada y se toma el penúltimo elemento.

```
numeros = (10, 25, 5, 40, 15)
segundo_mayor = sorted(numeros)[-2]
print("El segundo mayor es:", segundo_mayor)
```

Ejercicio 95

Enunciado: Crea una tupla con nombres y muéstralos en orden alfabético.
Explicación: Se usa `sorted()` para ordenar los nombres.

```
nombres = ("Carlos", "Ana", "Beatriz", "David")
ordenados = tuple(sorted(nombres))
print("Nombres ordenados:", ordenados)
```

Ejercicio 96

Enunciado: Crea una tupla con números y muestra solo los múltiplos de 3.
Explicación: Se filtran con `x % 3 == 0` dentro de una comprensión.

```
tupla = (3, 4, 9, 10, 12, 15, 20)
multiplos_3 = tuple(x for x in tupla if x % 3 == 0)
print("Múltiplos de 3:", multiplos_3)
```

Ejercicio 97

Enunciado: Crea una tupla con 5 palabras y muestra la más corta.
Explicación: Se usa `min()` con el argumento `key=len`.

```
palabras = ("sol", "montaña", "cielo", "río", "mar")
print("Palabra más corta:", min(palabras, key=len))
```

Ejercicio 98

Enunciado: Crea una tupla con nombres de animales y muestra cuántos tienen más de 5 letras.
Explicación: Se filtran con una comprensión y se usa `len()` para contar.

```
animales = ("perro", "gato", "elefante", "ratón", "jirafa")
largos = tuple(a for a in animales if len(a) > 5)
print("Animales con más de 5 letras:", largos)
```

Ejercicio 99

Enunciado: Crea una tupla con números y calcula la suma de los elementos en posiciones pares.
Explicación: Se usa `range(0, len(tupla), 2)` para recorrer solo los índices pares.

```
tupla = (10, 20, 30, 40, 50, 60)
suma_pares = sum(tupla[i] for i in range(0, len(tupla), 2))
print("Suma de posiciones pares:", suma_pares)
```

Ejercicio 100

Enunciado: Crea una tupla con 10 números y muestra los valores en orden inverso.
Explicación: Se usa la notación `[::-1]` para invertir la tupla.

```
numeros = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
inversa = numeros[::-1]
print("Tupla invertida:", inversa)
```

Ejercicios python diccionarios

Ejercicio 1

Enunciado: Crea un diccionario con los nombres de tres personas y sus edades.
Explicación: Un diccionario almacena pares `clave:valor`.

```
personas = {"Ana": 25, "Luis": 30, "Marta": 22}
print(personas)
```

Ejercicio 2

Enunciado: Muestra la edad de "Luis" en el diccionario anterior.
Explicación: Se accede a un valor usando su clave.

```
personas = {"Ana": 25, "Luis": 30, "Marta": 22}
print("Edad de Luis:", personas["Luis"])
```

Ejercicio 3

Enunciado: Añade una nueva persona al diccionario.
Explicación: Se asigna un valor a una nueva clave.

```
personas = {"Ana": 25, "Luis": 30, "Marta": 22}
personas["Pedro"] = 28
print(personas)
```

Ejercicio 4

Enunciado: Modifica la edad de "Ana".
Explicación: Se asigna un nuevo valor a una clave existente.

```
personas = {"Ana": 25, "Luis": 30, "Marta": 22}
personas["Ana"] = 26
print(personas)
```

Ejercicio 5

Enunciado: Elimina a "Marta" del diccionario.
Explicación: Se usa `del` para eliminar un par `clave:valor`.

```
personas = {"Ana": 25, "Luis": 30, "Marta": 22}
del personas["Marta"]
print(personas)
```

Ejercicio 6

Enunciado: Muestra todas las claves del diccionario.
Explicación: Se usa el método `.keys()` para obtener solo las claves.

```
personas = {"Ana": 25, "Luis": 30, "Marta": 22}
print("Claves:", personas.keys())
```

Ejercicio 7

Enunciado: Muestra todos los valores del diccionario.
Explicación: Se usa el método `.values()` para obtener solo los valores.

```
personas = {"Ana": 25, "Luis": 30, "Marta": 22}
print("Valores:", personas.values())
```

Ejercicio 8

Enunciado: Muestra todos los pares clave-valor del diccionario.
Explicación: Se usa el método `.items()` para obtener una lista de tuplas `(clave, valor)`.

```
personas = {"Ana": 25, "Luis": 30, "Marta": 22}
print("Pares clave-valor:", personas.items())
```

Ejercicio 9

Enunciado: Comprueba si "Pedro" está en el diccionario.
Explicación: Se usa el operador `in` para verificar la existencia de una clave.

```
personas = {"Ana": 25, "Luis": 30, "Marta": 22}
print("¿Pedro está en el diccionario?", "Pedro" in personas)
```

Ejercicio 10

Enunciado: Obtén la edad de "Marta" usando el método `.get()` para evitar errores.
Explicación: `.get()` devuelve `None` si la clave no existe, evitando un `KeyError`.

```
personas = {"Ana": 25, "Luis": 30, "Marta": 22}
edad_marta = personas.get("Marta")
print("Edad de Marta:", edad_marta)
```

Ejercicio 11

Enunciado: Crea un diccionario con frutas y sus precios.
Explicación: Un diccionario permite asociar cada fruta con su precio.

```
frutas = {"manzana": 1.5, "pera": 2.0, "plátano": 1.2}
print(frutas)
```

Ejercicio 12

Enunciado: Muestra el precio de la pera.
Explicación: Se accede al valor usando la clave correspondiente.

```
frutas = {"manzana": 1.5, "pera": 2.0, "plátano": 1.2}
print("Precio de la pera:", frutas["pera"])
```

Ejercicio 13

Enunciado: Cambia el precio del plátano a 1.5.
Explicación: Se modifica el valor de una clave existente.

```
frutas = {"manzana": 1.5, "pera": 2.0, "plátano": 1.2}
frutas["plátano"] = 1.5
print(frutas)
```

Ejercicio 14

Enunciado: Añade una nueva fruta con su precio.
Explicación: Se asigna un valor a una nueva clave en el diccionario.

```
frutas = {"manzana": 1.5, "pera": 2.0, "plátano": 1.5}
frutas["naranja"] = 1.8
print(frutas)
```

Ejercicio 15

Enunciado: Elimina la manzana del diccionario.
Explicación: Se usa `del` para eliminar un par `clave:valor`.

```
frutas = {"manzana": 1.5, "pera": 2.0, "plátano": 1.5, "naranja":
1.8}
del frutas["manzana"]
print(frutas)
```

Ejercicio 16

Enunciado: Muestra todas las claves del diccionario de frutas.
Explicación: `.keys()` devuelve un objeto con todas las claves.

```
frutas = {"pera": 2.0, "plátano": 1.5, "naranja": 1.8}
print("Claves:", frutas.keys())
```

Ejercicio 17

Enunciado: Muestra todos los valores del diccionario de frutas.
Explicación: `.values()` devuelve un objeto con todos los valores.

```
frutas = {"pera": 2.0, "plátano": 1.5, "naranja": 1.8}
print("Valores:", frutas.values())
```

Ejercicio 18

Enunciado: Muestra todos los pares clave-valor.
Explicación: `.items()` devuelve una lista de tuplas `(clave, valor)`.

```
frutas = {"pera": 2.0, "plátano": 1.5, "naranja": 1.8}
print("Pares clave-valor:", frutas.items())
```

Ejercicio 19

Enunciado: Comprueba si “mango” está en el diccionario.
Explicación: Se usa `in` para verificar si una clave existe.

```
frutas = {"pera": 2.0, "plátano": 1.5, "naranja": 1.8}
print("¿Mango está en el diccionario?", "mango" in frutas)
```

Ejercicio 20

Enunciado: Obtén el precio de la naranja usando `.get()` para evitar errores.
Explicación: `.get()` devuelve `None` si la clave no existe, evitando un error.

```
frutas = {"pera": 2.0, "plátano": 1.5, "naranja": 1.8}
precio_naranja = frutas.get("naranja")
print("Precio de la naranja:", precio_naranja)
```

Ejercicio 21

Enunciado: Crea un diccionario con nombres de estudiantes y sus calificaciones.
Explicación: Un diccionario asocia cada estudiante con su calificación.

```
calificaciones = {"Ana": 8, "Luis": 9, "Marta": 7}
print(calificaciones)
```

Ejercicio 22

Enunciado: Muestra la calificación de "Marta".
Explicación: Se accede a un valor usando la clave correspondiente.

```
calificaciones = {"Ana": 8, "Luis": 9, "Marta": 7}
print("Calificación de Marta:", calificaciones["Marta"])
```

Ejercicio 23

Enunciado: Modifica la calificación de "Ana" a 10.
Explicación: Se asigna un nuevo valor a una clave existente.

```
calificaciones = {"Ana": 8, "Luis": 9, "Marta": 7}
calificaciones["Ana"] = 10
print(calificaciones)
```

Ejercicio 24

Enunciado: Añade un nuevo estudiante al diccionario.
Explicación: Se crea una nueva clave con su valor asociado.

```
calificaciones = {"Ana": 10, "Luis": 9, "Marta": 7}
calificaciones["Pedro"] = 8
print(calificaciones)
```

Ejercicio 25

Enunciado: Elimina a "Luis" del diccionario.
Explicación: Se usa `del` para eliminar un par clave-valor.

```
calificaciones = {"Ana": 10, "Luis": 9, "Marta": 7, "Pedro": 8}
del calificaciones["Luis"]
print(calificaciones)
```

Ejercicio 26

Enunciado: Muestra todas las claves del diccionario de calificaciones.
Explicación: `.keys()` devuelve un objeto con todas las claves del diccionario.

```
calificaciones = {"Ana": 10, "Marta": 7, "Pedro": 8}
print("Claves:", calificaciones.keys())
```

Ejercicio 27

Enunciado: Muestra todos los valores del diccionario de calificaciones.
Explicación: `.values()` devuelve un objeto con todos los valores.

```
calificaciones = {"Ana": 10, "Marta": 7, "Pedro": 8}
print("Valores:", calificaciones.values())
```

Ejercicio 28

Enunciado: Muestra todos los pares clave-valor.
Explicación: `.items()` devuelve una lista de tuplas `(clave, valor)`.

```
calificaciones = {"Ana": 10, "Marta": 7, "Pedro": 8}
print("Pares clave-valor:", calificaciones.items())
```

Ejercicio 29

Enunciado: Comprueba si "Luis" está en el diccionario.
Explicación: Se usa `in` para verificar la existencia de una clave.

```
calificaciones = {"Ana": 10, "Marta": 7, "Pedro": 8}
print("¿Luis está en el diccionario?", "Luis" in calificaciones)
```

Ejercicio 30

Enunciado: Obtén la calificación de "Pedro" usando `.get()` para evitar errores si no existe la clave.
Explicación: `.get()` devuelve `None` si la clave no se encuentra.

```
calificaciones = {"Ana": 10, "Marta": 7, "Pedro": 8}
calificacion_pedro = calificaciones.get("Pedro")
print("Calificación de Pedro:", calificacion_pedro)
```

Ejercicio 31

Enunciado: Crea un diccionario con nombres de ciudades y su población.
Explicación: Un diccionario permite asociar cada ciudad con su población.

```
ciudades = {"Madrid": 3200000, "Barcelona": 1600000, "Valencia":
800000}
print(ciudades)
```

Ejercicio 32

Enunciado: Muestra la población de Barcelona.
Explicación: Se accede al valor usando la clave correspondiente.

```
ciudades = {"Madrid": 3200000, "Barcelona": 1600000, "Valencia":
800000}
print("Población de Barcelona:", ciudades["Barcelona"])
```

Ejercicio 33

Enunciado: Cambia la población de Valencia a 810000.
Explicación: Se modifica el valor de una clave existente.

```
ciudades = {"Madrid": 3200000, "Barcelona": 1600000, "Valencia":
800000}
ciudades["Valencia"] = 810000
print(ciudades)
```

Ejercicio 34

Enunciado: Añade una nueva ciudad con su población.
Explicación: Se agrega una nueva clave con su valor asociado.

```
ciudades = {"Madrid": 3200000, "Barcelona": 1600000, "Valencia":
810000}
ciudades["Sevilla"] = 700000
print(ciudades)
```

Ejercicio 35

Enunciado: Elimina Madrid del diccionario.
Explicación: Se usa `del` para eliminar un par clave-valor.

```
ciudades = {"Madrid": 3200000, "Barcelona": 1600000, "Valencia":
810000, "Sevilla": 700000}
del ciudades["Madrid"]
print(ciudades)
```

Ejercicio 36

Enunciado: Muestra todas las claves del diccionario de ciudades.
Explicación: `.keys()` devuelve un objeto con todas las claves.

```
ciudades = {"Barcelona": 1600000, "Valencia": 810000, "Sevilla":
700000}
print("Claves:", ciudades.keys())
```

Ejercicio 37

Enunciado: Muestra todos los valores del diccionario de ciudades.
Explicación: `.values()` devuelve un objeto con todos los valores.

```
ciudades = {"Barcelona": 1600000, "Valencia": 810000, "Sevilla":
700000}
print("Valores:", ciudades.values())
```

Ejercicio 38

Enunciado: Muestra todos los pares clave-valor.
Explicación: `.items()` devuelve una lista de tuplas `(clave, valor)`.

```
ciudades = {"Barcelona": 1600000, "Valencia": 810000, "Sevilla":
700000}
print("Pares clave-valor:", ciudades.items())
```

Ejercicio 39

Enunciado: Comprueba si “Bilbao” está en el diccionario.
Explicación: Se usa `in` para verificar la existencia de una clave.

```
ciudades = {"Barcelona": 1600000, "Valencia": 810000, "Sevilla":
700000}
print("¿Bilbao está en el diccionario?", "Bilbao" in ciudades)
```

Ejercicio 40

Enunciado: Obtén la población de Sevilla usando `.get()`.

Explicación: `.get()` devuelve `None` si la clave no existe, evitando errores.

```
ciudades = {"Barcelona": 1600000, "Valencia": 810000, "Sevilla":
700000}
poblacion_sevilla = ciudades.get("Sevilla")
print("Población de Sevilla:", poblacion_sevilla)
```

Ejercicio 41

Enunciado: Crea un diccionario con asignaturas y notas de un estudiante.

Explicación: Cada asignatura se asocia con su nota usando un diccionario.

```
notas = {"Matemáticas": 8, "Física": 7, "Química": 9}
print(notas)
```

Ejercicio 42

Enunciado: Muestra la nota de Física.

Explicación: Se accede al valor de la clave correspondiente.

```
notas = {"Matemáticas": 8, "Física": 7, "Química": 9}
print("Nota de Física:", notas["Física"])
```

Ejercicio 43

Enunciado: Modifica la nota de Química a 10.

Explicación: Se asigna un nuevo valor a la clave existente.

```
notas = {"Matemáticas": 8, "Física": 7, "Química": 9}
notas["Química"] = 10
print(notas)
```

Ejercicio 44

Enunciado: Añade una nueva asignatura con su nota.

Explicación: Se crea una nueva clave con su valor.

```
notas = {"Matemáticas": 8, "Física": 7, "Química": 10}
notas["Biología"] = 9
print(notas)
```

Ejercicio 45

Enunciado: Elimina Matemáticas del diccionario.
Explicación: Se usa `del` para eliminar un par clave-valor.

```
notas = {"Matemáticas": 8, "Física": 7, "Química": 10, "Biología": 9}
del notas["Matemáticas"]
print(notas)
```

Ejercicio 46

Enunciado: Muestra todas las asignaturas disponibles.
Explicación: `.keys()` devuelve todas las claves del diccionario.

```
notas = {"Física": 7, "Química": 10, "Biología": 9}
print("Asignaturas:", notas.keys())
```

Ejercicio 47

Enunciado: Muestra todas las notas de las asignaturas.
Explicación: `.values()` devuelve todos los valores del diccionario.

```
notas = {"Física": 7, "Química": 10, "Biología": 9}
print("Notas:", notas.values())
```

Ejercicio 48

Enunciado: Muestra todos los pares asignatura-nota.
Explicación: `.items()` devuelve una lista de tuplas `(clave, valor)`.

```
notas = {"Física": 7, "Química": 10, "Biología": 9}
print("Asignaturas y notas:", notas.items())
```

Ejercicio 49

Enunciado: Comprueba si “Historia” está en el diccionario.
Explicación: Se usa `in` para verificar la existencia de una clave.

```
notas = {"Física": 7, "Química": 10, "Biología": 9}
print("¿Historia está en el diccionario?", "Historia" in notas)
```

Ejercicio 50

Enunciado: Obtén la nota de Biología usando `.get()`.
Explicación: `.get()` devuelve `None` si la clave no existe, evitando errores.

```
notas = {"Física": 7, "Química": 10, "Biología": 9}
nota_biologia = notas.get("Biología")
print("Nota de Biología:", nota_biologia)
```

Ejercicio 51

Enunciado: Crea un diccionario con productos y sus cantidades en stock.
Explicación: Cada producto se asocia con su cantidad disponible.

```
productos = {"leche": 20, "pan": 50, "huevos": 30}

print(productos)
```

Ejercicio 52

Enunciado: Muestra la cantidad de pan en stock.
Explicación: Se accede al valor usando la clave correspondiente.

```
productos = {"leche": 20, "pan": 50, "huevos": 30}

print("Cantidad de pan:", productos["pan"])
```

Ejercicio 53

Enunciado: Modifica la cantidad de huevos a 25.
Explicación: Se asigna un nuevo valor a una clave existente.

```
productos = {"leche": 20, "pan": 50, "huevos": 30}

productos["huevos"] = 25

print(productos)
```

Ejercicio 54

Enunciado: Añade un nuevo producto con su cantidad.
Explicación: Se agrega un nuevo par clave-valor al diccionario.

```
productos = {"leche": 20, "pan": 50, "huevos": 25}

productos["mantequilla"] = 15

print(productos)
```

Ejercicio 55

Enunciado: Elimina la leche del diccionario.
Explicación: Se usa `del` para eliminar un par clave-valor.

```
productos = {"leche": 20, "pan": 50, "huevos": 25, "mantequilla": 15}
del productos["leche"]
print(productos)
```

Ejercicio 56

Enunciado: Muestra todas las claves del diccionario de productos.
Explicación: `.keys()` devuelve un objeto con todas las claves del diccionario.

```
productos = {"pan": 50, "huevos": 25, "mantequilla": 15}
print("Productos:", productos.keys())
```

Ejercicio 57

Enunciado: Muestra todos los valores del diccionario de productos.
Explicación: `.values()` devuelve todos los valores del diccionario.

```
productos = {"pan": 50, "huevos": 25, "mantequilla": 15}
print("Cantidades:", productos.values())
```

Ejercicio 58

Enunciado: Muestra todos los pares producto-cantidad.
Explicación: `.items()` devuelve una lista de tuplas `(clave, valor)`.

```
productos = {"pan": 50, "huevos": 25, "mantequilla": 15}
print("Productos y cantidades:", productos.items())
```

Ejercicio 59

Enunciado: Comprueba si “azúcar” está en el diccionario.
Explicación: Se usa `in` para verificar la existencia de una clave.

```
productos = {"pan": 50, "huevos": 25, "mantequilla": 15}
print("¿Azúcar está en el diccionario?", "azúcar" in productos)
```

Ejercicio 60

Enunciado: Obtén la cantidad de mantequilla usando `.get()`.
Explicación: `.get()` devuelve `None` si la clave no existe, evitando errores.

```
productos = {"pan": 50, "huevos": 25, "mantequilla": 15}

cantidad_mantequilla = productos.get("mantequilla")

print("Cantidad de mantequilla:", cantidad_mantequilla)
```

Ejercicio 61

Enunciado: Crea un diccionario con países y sus capitales.
Explicación: Cada país se asocia con su capital usando un diccionario.

```
capitales = {"España": "Madrid", "Francia": "París", "Italia":
"Roma"}
print(capitales)
```

Ejercicio 62

Enunciado: Muestra la capital de Francia.
Explicación: Se accede al valor usando la clave correspondiente.

```
capitales = {"España": "Madrid", "Francia": "París", "Italia":
"Roma"}
print("Capital de Francia:", capitales["Francia"])
```

Ejercicio 63

Enunciado: Cambia la capital de Italia a "Milán".
Explicación: Se asigna un nuevo valor a la clave existente.

```
capitales = {"España": "Madrid", "Francia": "París", "Italia":
"Roma"}
capitales["Italia"] = "Milán"
print(capitales)
```

Ejercicio 64

Enunciado: Añade un nuevo país con su capital.
Explicación: Se agrega un nuevo par clave-valor al diccionario.

```
capitales = {"España": "Madrid", "Francia": "París", "Italia":
"Milán"}
capitales["Alemania"] = "Berlín"
print(capitales)
```

Ejercicio 65

Enunciado: Elimina España del diccionario.
Explicación: Se usa `del` para eliminar un par clave-valor.

```
capitales = {"España": "Madrid", "Francia": "París", "Italia":
"Milán", "Alemania": "Berlín"}
del capitales["España"]
print(capitales)
```

Ejercicio 66

Enunciado: Muestra todas las claves del diccionario de países.
Explicación: `.keys()` devuelve un objeto con todas las claves.

```
capitales = {"Francia": "París", "Italia": "Milán", "Alemania":
"Berlín"}
print("Países:", capitales.keys())
```

Ejercicio 67

Enunciado: Muestra todas las capitales del diccionario.
Explicación: `.values()` devuelve todos los valores del diccionario.

```
capitales = {"Francia": "París", "Italia": "Milán", "Alemania":
"Berlín"}
print("Capitales:", capitales.values())
```

Ejercicio 68

Enunciado: Muestra todos los pares país-capital.
Explicación: `.items()` devuelve una lista de tuplas `(clave, valor)`.

```
capitales = {"Francia": "París", "Italia": "Milán", "Alemania":
"Berlín"}
print("Países y capitales:", capitales.items())
```

Ejercicio 69

Enunciado: Comprueba si “Portugal” está en el diccionario.
Explicación: Se usa `in` para verificar la existencia de una clave.

```
capitales = {"Francia": "París", "Italia": "Milán", "Alemania":
"Berlín"}
print("¿Portugal está en el diccionario?", "Portugal" in
capitales)
```

Ejercicio 70

Enunciado: Obtén la capital de Alemania usando `.get()`.
Explicación: `.get()` devuelve `None` si la clave no existe, evitando errores.

```
capitales = {"Francia": "París", "Italia": "Milán", "Alemania":
"Berlín"}
capital_alemania = capitales.get("Alemania")
print("Capital de Alemania:", capital_alemania)
```

Ejercicio 71

Enunciado: Crea un diccionario con empleados y sus salarios.
Explicación: Cada empleado se asocia con su salario usando un diccionario.

```
empleados = {"Ana": 2000, "Luis": 2500, "Marta": 1800}

print(empleados)
```

Ejercicio 72

Enunciado: Muestra el salario de Luis.
Explicación: Se accede al valor usando la clave correspondiente.

```
empleados = {"Ana": 2000, "Luis": 2500, "Marta": 1800}

print("Salario de Luis:", empleados["Luis"])
```

Ejercicio 73

Enunciado: Cambia el salario de Ana a 2200.
Explicación: Se asigna un nuevo valor a una clave existente.

```
empleados = {"Ana": 2000, "Luis": 2500, "Marta": 1800}

empleados["Ana"] = 2200

print(empleados)
```

Ejercicio 74

Enunciado: Añade un nuevo empleado con su salario.
Explicación: Se agrega un nuevo par clave-valor al diccionario.

```
empleados = {"Ana": 2200, "Luis": 2500, "Marta": 1800}

empleados["Pedro"] = 2100

print(empleados)
```

Ejercicio 75

Enunciado: Elimina a Marta del diccionario.
Explicación: Se usa `del` para eliminar un par clave-valor.

```
empleados = {"Ana": 2200, "Luis": 2500, "Marta": 1800, "Pedro": 2100}

del empleados["Marta"]

print(empleados)
```

Ejercicio 76

Enunciado: Muestra todos los nombres de empleados.
Explicación: `.keys()` devuelve todas las claves del diccionario.

```
empleados = {"Ana": 2200, "Luis": 2500, "Pedro": 2100}

print("Empleados:", empleados.keys())
```

Ejercicio 77

Enunciado: Muestra todos los salarios de los empleados.
Explicación: `.values()` devuelve todos los valores del diccionario.

```
empleados = {"Ana": 2200, "Luis": 2500, "Pedro": 2100}

print("Salarios:", empleados.values())
```

Ejercicio 78

Enunciado: Muestra todos los pares empleado-salario.
Explicación: `.items()` devuelve una lista de tuplas `(clave, valor)`.

```
empleados = {"Ana": 2200, "Luis": 2500, "Pedro": 2100}

print("Empleados y salarios:", empleados.items())
```

Ejercicio 79

Enunciado: Comprueba si “Marta” está en el diccionario.
Explicación: Se usa `in` para verificar la existencia de una clave.

```
empleados = {"Ana": 2200, "Luis": 2500, "Pedro": 2100}
print("¿Marta está en el diccionario?", "Marta" in empleados)
```

Ejercicio 80

Enunciado: Obtén el salario de Pedro usando `.get()`.
Explicación: `.get()` devuelve `None` si la clave no existe, evitando errores.

```
empleados = {"Ana": 2200, "Luis": 2500, "Pedro": 2100}
salario_pedro = empleados.get("Pedro")
print("Salario de Pedro:", salario_pedro)
```

Ejercicio 81

Enunciado: Crea un diccionario con coches y sus colores.
Explicación: Cada coche se asocia con su color usando un diccionario.

```
coches = {"Ford": "Rojo", "Toyota": "Azul", "BMW": "Negro"}
print(coches)
```

Ejercicio 82

Enunciado: Muestra el color del Toyota.
Explicación: Se accede al valor usando la clave correspondiente.

```
coches = {"Ford": "Rojo", "Toyota": "Azul", "BMW": "Negro"}
print("Color del Toyota:", coches["Toyota"])
```

Ejercicio 83

Enunciado: Cambia el color del BMW a “Blanco”.
Explicación: Se asigna un nuevo valor a una clave existente.

```
coches = {"Ford": "Rojo", "Toyota": "Azul", "BMW": "Negro"}
coches["BMW"] = "Blanco"
print(coches)
```

Ejercicio 84

Enunciado: Añade un nuevo coche con su color.
Explicación: Se agrega un nuevo par clave-valor al diccionario.

```
coches = {"Ford": "Rojo", "Toyota": "Azul", "BMW": "Blanco"}
coches["Honda"] = "Verde"
print(coches)
```

Ejercicio 85

Enunciado: Elimina el Ford del diccionario.
Explicación: Se usa `del` para eliminar un par clave-valor.

```
coches = {"Ford": "Rojo", "Toyota": "Azul", "BMW": "Blanco", "Honda": "Verde"}
del coches["Ford"]
print(coches)
```

Ejercicio 86

Enunciado: Muestra todos los coches disponibles.
Explicación: `.keys()` devuelve todas las claves del diccionario.

```
coches = {"Toyota": "Azul", "BMW": "Blanco", "Honda": "Verde"}
print("Coches:", coches.keys())
```

Ejercicio 87

Enunciado: Muestra todos los colores de los coches.
Explicación: `.values()` devuelve todos los valores del diccionario.

```
coches = {"Toyota": "Azul", "BMW": "Blanco", "Honda": "Verde"}
print("Colores:", coches.values())
```

Ejercicio 88

Enunciado: Muestra todos los pares coche-color.
Explicación: `.items()` devuelve una lista de tuplas `(clave, valor)`.

```
coches = {"Toyota": "Azul", "BMW": "Blanco", "Honda": "Verde"}
print("Coches y colores:", coches.items())
```

Ejercicio 89

Enunciado: Comprueba si “Ford” está en el diccionario.
Explicación: Se usa `in` para verificar la existencia de una clave.

```
coches = {"Toyota": "Azul", "BMW": "Blanco", "Honda": "Verde"}

print("¿Ford está en el diccionario?", "Ford" in coches)
```

Ejercicio 90

Enunciado: Obtén el color del Honda usando `.get()`.
Explicación: `.get()` devuelve `None` si la clave no existe, evitando errores.

```
coches = {"Toyota": "Azul", "BMW": "Blanco", "Honda": "Verde"}

color_honda = coches.get("Honda")

print("Color del Honda:", color_honda)
```

Ejercicio 91

Enunciado: Crea un diccionario con nombres de artistas y su género musical.
Explicación: Cada artista se asocia con su género musical usando un diccionario.

```
artistas = {"Adele": "Pop", "Metallica": "Rock", "Mozart":
"Clásica"}

print(artistas)
```

Ejercicio 92

Enunciado: Muestra el género musical de Metallica.
Explicación: Se accede al valor usando la clave correspondiente.

```
artistas = {"Adele": "Pop", "Metallica": "Rock", "Mozart":
"Clásica"}

print("Género de Metallica:", artistas["Metallica"])
```

Ejercicio 93

Enunciado: Cambia el género musical de Adele a “Soul”.
Explicación: Se asigna un nuevo valor a una clave existente.

```
artistas = {"Adele": "Pop", "Metallica": "Rock", "Mozart":
"Clásica"}

artistas["Adele"] = "Soul"

print(artistas)
```

Ejercicio 94

Enunciado: Añade un nuevo artista con su género musical.
Explicación: Se agrega un nuevo par clave-valor al diccionario.

```
artistas = {"Adele": "Soul", "Metallica": "Rock", "Mozart": "Clásica"}

artistas["Shakira"] = "Pop"

print(artistas)
```

Ejercicio 95

Enunciado: Elimina a Mozart del diccionario.
Explicación: Se usa `del` para eliminar un par clave-valor.

```
artistas = {"Adele": "Soul", "Metallica": "Rock", "Mozart": "Clásica", "Shakira": "Pop"}

del artistas["Mozart"]

print(artistas)
```

Ejercicio 96

Enunciado: Muestra todos los artistas disponibles.
Explicación: `.keys()` devuelve todas las claves del diccionario.

```
artistas = {"Adele": "Soul", "Metallica": "Rock", "Shakira": "Pop"}

print("Artistas:", artistas.keys())
```

Ejercicio 97

Enunciado: Muestra todos los géneros musicales del diccionario.
Explicación: `.values()` devuelve todos los valores del diccionario.

```
artistas = {"Adele": "Soul", "Metallica": "Rock", "Shakira": "Pop"}

print("Géneros musicales:", artistas.values())
```

Ejercicio 98

Enunciado: Muestra todos los pares artista-género.
Explicación: `.items()` devuelve una lista de tuplas `(clave, valor)`.

```
artistas = {"Adele": "Soul", "Metallica": "Rock", "Shakira":
"Pop"}

print("Artistas y géneros:", artistas.items())
```

Ejercicio 99

Enunciado: Comprueba si “Mozart” está en el diccionario.
Explicación: Se usa `in` para verificar la existencia de una clave.

```
artistas = {"Adele": "Soul", "Metallica": "Rock", "Shakira":
"Pop"}

print("¿Mozart está en el diccionario?", "Mozart" in artistas)
```

Ejercicio 100

Enunciado: Obtén el género musical de Shakira usando `.get()`.
Explicación: `.get()` devuelve `None` si la clave no existe, evitando errores.

```
artistas = {"Adele": "Soul", "Metallica": "Rock", "Shakira":
"Pop"}

genero_shakira = artistas.get("Shakira")

print("Género de Shakira:", genero_shakira)
```

Ejercicios python funciones

Ejercicio 1

Enunciado: Crea una función que imprima “Hola mundo”.
Explicación: Las funciones permiten agrupar código reutilizable. Esta función no recibe parámetros ni devuelve valores.

```
def saludar():
    print("Hola mundo")

saludar()
```

Ejercicio 2

Enunciado: Crea una función que reciba un nombre y lo imprima.
Explicación: La función recibe un parámetro y lo utiliza dentro de su bloque de código.

```
def saludar(nombre):
    print("Hola", nombre)

saludar("Ana")
```

Ejercicio 3

Enunciado: Crea una función que sume dos números y muestre el resultado.
Explicación: La función recibe dos parámetros, realiza la operación y la imprime.

```
def sumar(a, b):
    resultado = a + b
    print("Suma:", resultado)

sumar(5, 3)
```

Ejercicio 4

Enunciado: Crea una función que devuelva el cuadrado de un número.
Explicación: La función usa `return` para devolver un valor calculado.

```
def cuadrado(numero):
    return numero ** 2

print(cuadrado(4))
```

Ejercicio 5

Enunciado: Crea una función que reciba dos números y devuelva su multiplicación.
Explicación: Permite reutilizar la operación de multiplicación con distintos valores.

```
def multiplicar(a, b):
    return a * b

print(multiplicar(5, 6))
```

Ejercicio 6

Enunciado: Crea una función que convierta grados Celsius a Fahrenheit.
Explicación: La función recibe un valor y devuelve el valor convertido usando una fórmula.

```
def celsius_a_fahrenheit(c):
    return (c * 9/5) + 32

print(celsius_a_fahrenheit(25))
```

Ejercicio 7

Enunciado: Crea una función que imprima los números del 1 al 5.
Explicación: La función puede contener bucles para repetir acciones.

```
def imprimir_numeros():
    for i in range(1, 6):
        print(i)

imprimir_numeros()
```

Ejercicio 8

Enunciado: Crea una función que reciba una lista y muestre sus elementos.
Explicación: La función puede trabajar con estructuras de datos pasadas como parámetros.

```
def mostrar_lista(lista):
    for elemento in lista:
        print(elemento)

mostrar_lista([1, 2, 3, 4])
```

Ejercicio 9

Enunciado: Crea una función que reciba un número y diga si es par o impar.
Explicación: La función puede tomar decisiones usando condicionales y devolver o imprimir resultados.

```
def par_o_impar(n):
    if n % 2 == 0:
        print(n, "es par")
    else:
        print(n, "es impar")

par_o_impar(7)
```

Ejercicio 10

Enunciado: Crea una función que devuelva el número mayor entre dos números.
Explicación: La función compara los valores y devuelve el mayor usando `return`.

```
def mayor(a, b):
    if a > b:
        return a
    else:
        return b

print(mayor(10, 20))
```

Ejercicio 11

Enunciado: Crea una función que reciba un número y devuelva su valor absoluto.
Explicación: `abs()` devuelve el valor absoluto; la función permite reutilizar este cálculo.

```
def valor_absoluto(n):
    return abs(n)


print(valor_absoluto(-7))
```

Ejercicio 12

Enunciado: Crea una función que reciba tres números y devuelva el promedio.
Explicación: La función permite agrupar la operación matemática y devolver el resultado.

```
def promedio(a, b, c):
    return (a + b + c) / 3


print(promedio(4, 5, 6))
```

Ejercicio 13

Enunciado: Crea una función que imprima “Hola” y el nombre pasado por parámetro.
Explicación: La función recibe un argumento y lo combina con texto.

```
def saludar_persona(nombre):
    print("Hola", nombre)


saludar_persona("Luis")
```

Ejercicio 14

Enunciado: Crea una función que reciba un número y devuelva True si es positivo, False si es negativo.
Explicación: La función permite evaluar condiciones y devolver un valor booleano.

```
def es_positivo(n):
    return n > 0


print(es_positivo(-3))
print(es_positivo(5))
```

Ejercicio 15

Enunciado: Crea una función que reciba un texto y lo imprima en mayúsculas.
Explicación: La función usa métodos de strings para transformar el texto.

```
def en_mayusculas(texto):
    print(texto.upper())


en_mayusculas("hola mundo")
```

Ejercicio 16

Enunciado: Crea una función que reciba un número y diga si es divisible por 3.
Explicación: La función permite evaluar condiciones y mostrar resultados.

```
def divisible_por_tres(n):
    if n % 3 == 0:
        print(n, "es divisible por 3")
    else:
        print(n, "no es divisible por 3")

divisible_por_tres(9)
divisible_por_tres(10)
```

Ejercicio 17

Enunciado: Crea una función que reciba dos números y devuelva su resta.
Explicación: La función realiza la operación matemática y devuelve el resultado.

```
def restar(a, b):
    return a - b

print(restar(10, 4))
```

Ejercicio 18

Enunciado: Crea una función que reciba una lista y devuelva la suma de sus elementos.
Explicación: La función permite procesar listas y devolver resultados calculados.

```
def suma_lista(lista):
    return sum(lista)

print(suma_lista([1, 2, 3, 4]))
```

Ejercicio 19

Enunciado: Crea una función que reciba un número y devuelva su doble.
Explicación: La función realiza una operación simple y devuelve el resultado.

```
def doble(n):
    return n * 2

print(doble(7))
```

Ejercicio 20

Enunciado: Crea una función que reciba un nombre y un apellido y los imprima juntos.
Explicación: La función recibe varios parámetros y los combina para mostrar un resultado.

```
def nombre_completo(nombre, apellido):
    print(nombre, apellido)

nombre_completo("Ana", "García")
```

Ejercicio 21

Enunciado: Crea una función que reciba un número y devuelva su triple.
Explicación: La función permite calcular el triple de cualquier número recibido como parámetro.

```
def triple(n):
    return n * 3

print(triple(5))
```

Ejercicio 22

Enunciado: Crea una función que reciba dos números y devuelva el menor.
Explicación: La función compara dos valores y devuelve el menor usando condicionales.

```
def menor(a, b):
    if a < b:
        return a
    else:
        return b

print(menor(10, 20))
```

Ejercicio 23

Enunciado: Crea una función que reciba un número y diga si es positivo, negativo o cero.
Explicación: La función evalúa condiciones y muestra un mensaje según el caso.

```
def tipo_numero(n):
    if n > 0:
        print(n, "es positivo")
    elif n < 0:
        print(n, "es negativo")
    else:
        print(n, "es cero")

tipo_numero(5)
tipo_numero(-3)
tipo_numero(0)
```

Ejercicio 24

Enunciado: Crea una función que reciba una lista y devuelva su longitud.
Explicación: La función utiliza `len()` para calcular la cantidad de elementos de la lista.

```
def longitud_lista(lista):
    return len(lista)

print(longitud_lista([1, 2, 3, 4, 5]))
```

Ejercicio 25

Enunciado: Crea una función que reciba un número y devuelva su raíz cuadrada.
Explicación: La función utiliza el operador de potencia `**` para calcular la raíz cuadrada.

```
def raiz_cuadrada(n):
    return n ** 0.5

print(raiz_cuadrada(16))
```

Ejercicio 26

Enunciado: Crea una función que reciba un número y diga si es múltiplo de 5.
Explicación: La función evalúa la divisibilidad del número usando el operador `%`.

```
def multiplo_cinco(n):
    if n % 5 == 0:
        print(n, "es múltiplo de 5")
    else:
        print(n, "no es múltiplo de 5")

multiplo_cinco(25)
multiplo_cinco(7)
```

Ejercicio 27

Enunciado: Crea una función que reciba dos números y devuelva su división.
Explicación: La función devuelve el resultado de dividir los dos números, evitando errores de división por cero.

```
def dividir(a, b):
    if b != 0:
        return a / b
    else:
        return "Error: división por cero"

print(dividir(10, 2))
print(dividir(5, 0))
```

Ejercicio 28

Enunciado: Crea una función que reciba un texto y devuelva su longitud.
Explicación: La función puede recibir cadenas y calcular su número de caracteres usando `len()`.

```
def longitud_texto(texto):
    return len(texto)

print(longitud_texto("Hola mundo"))
```

Ejercicio 29

Enunciado: Crea una función que reciba una lista y devuelva el elemento mayor.
Explicación: La función utiliza `max()` para encontrar el valor máximo en una lista.

```
def max_lista(lista):
    return max(lista)

print(max_lista([3, 7, 1, 9, 5]))
```

Ejercicio 30

Enunciado: Crea una función que reciba un número y devuelva True si es par, False si es impar.
Explicación: La función evalúa la paridad de un número y devuelve un valor booleano.

```
def es_par(n):
    return n % 2 == 0

print(es_par(4))
print(es_par(7))
```

Ejercicio 31

Enunciado: Crea una función que reciba dos números y devuelva su suma y resta.
Explicación: La función puede devolver múltiples valores usando tuplas.

```
def suma_resta(a, b):
    return a + b, a - b


resultado_suma, resultado_resta = suma_resta(10, 5)
print("Suma:", resultado_suma, "Resta:", resultado_resta)
```

Ejercicio 32

Enunciado: Crea una función que reciba un número y devuelva True si es múltiplo de 2 y 3.
Explicación: Evalúa dos condiciones simultáneamente y devuelve un valor booleano.

```
def multiplo_dos_tres(n):
    return n % 2 == 0 and n % 3 == 0


print(multiplo_dos_tres(12))
print(multiplo_dos_tres(10))
```

Ejercicio 33

Enunciado: Crea una función que reciba una lista y devuelva el primer elemento.
Explicación: Permite acceder a elementos específicos de una lista usando índices.

```
def primer_elemento(lista):
    return lista[0] if lista else None


print(primer_elemento([5, 7, 9]))
```

Ejercicio 34

Enunciado: Crea una función que reciba un número y devuelva su valor al cubo.
Explicación: La función realiza una operación matemática simple y devuelve el resultado.

```
def cubo(n):
    return n ** 3


print(cubo(3))
```

Ejercicio 35

Enunciado: Crea una función que reciba un nombre y una edad y los imprima en formato frase.
Explicación: La función recibe múltiples parámetros y los combina para formar un mensaje.

```
def presentar(nombre, edad):
    print(f"Me llamo {nombre} y tengo {edad} años.")


presentar("Luis", 25)
```

Ejercicio 36

Enunciado: Crea una función que reciba una lista y devuelva el último elemento.
Explicación: Permite acceder al último elemento usando índices negativos.

```
def ultimo_elemento(lista):
    return lista[-1] if lista else None


print(ultimo_elemento([5, 7, 9]))
```

Ejercicio 37

Enunciado: Crea una función que reciba un número y devuelva True si es mayor que 10.
Explicación: Evalúa una condición simple y devuelve un valor booleano.

```
def mayor_10(n):
    return n > 10


print(mayor_10(15))
print(mayor_10(7))
```

Ejercicio 38

Enunciado: Crea una función que reciba un texto y lo devuelva invertido.
Explicación: La función permite manipular strings usando slicing.

```
def invertir_texto(texto):
    return texto[::-1]


print(invertir_texto("Python"))
```

Ejercicio 39

Enunciado: Crea una función que reciba dos números y devuelva True si son iguales.
Explicación: Evalúa la igualdad de dos valores y devuelve un booleano.

```
def son_iguales(a, b):
    return a == b


print(son_iguales(5, 5))
print(son_iguales(5, 3))
```

Ejercicio 40

Enunciado: Crea una función que reciba una lista y devuelva la suma de sus elementos pares.
Explicación: La función recorre la lista, filtra los pares y suma sus valores.

```
def suma_pares(lista):
    return sum(x for x in lista if x % 2 == 0)


print(suma_pares([1, 2, 3, 4, 5, 6]))
```

Ejercicio 41

Enunciado: Crea una función que reciba un número y devuelva su valor absoluto si es negativo, o el mismo número si es positivo.
Explicación: La función utiliza una condición para devolver el valor absoluto solo si es necesario.

```
def absoluto_condicional(n):
    if n < 0:
        return -n
    return n

print(absoluto_condicional(-8))
print(absoluto_condicional(5))
```

Ejercicio 42

Enunciado: Crea una función que reciba una lista y devuelva la lista invertida.
Explicación: La función manipula listas usando slicing para invertir el orden de sus elementos.

```
def invertir_lista(lista):
    return lista[::-1]

print(invertir_lista([1, 2, 3, 4]))
```

Ejercicio 43

Enunciado: Crea una función que reciba un número y devuelva True si es múltiplo de 4, False si no.
Explicación: Evalúa la divisibilidad usando el operador % y devuelve un valor booleano.

```
def multiplo_cuatro(n):
    return n % 4 == 0

print(multiplo_cuatro(8))
print(multiplo_cuatro(7))
```

Ejercicio 44

Enunciado: Crea una función que reciba un nombre y lo imprima en mayúsculas y minúsculas.
Explicación: La función permite manipular strings y mostrar diferentes formatos.

```
def mostrar_mayus_minus(nombre):
    print(nombre.upper())
    print(nombre.lower())

mostrar_mayus_minus("Ana")
```

Ejercicio 45

Enunciado: Crea una función que reciba un número y devuelva su factorial.
Explicación: La función utiliza un bucle `for` para calcular el factorial de manera sencilla.

```
def factorial(n):
    resultado = 1
    for i in range(1, n + 1):
        resultado *= i
    return resultado

print(factorial(5))
```

Ejercicio 46

Enunciado: Crea una función que reciba dos números y devuelva True si el primero es divisible por el segundo.
Explicación: La función evalúa la divisibilidad usando `%` y devuelve un booleano.

```
def divisible(a, b):
    return a % b == 0

print(divisible(10, 2))
print(divisible(7, 3))
```

Ejercicio 47

Enunciado: Crea una función que reciba un número y diga si es primo o no.
Explicación: La función recorre los posibles divisores para determinar si un número es primo.

```
def es_primo(n):
    if n < 2:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

print(es_primo(7))
print(es_primo(10))
```

Ejercicio 48

Enunciado: Crea una función que reciba una lista y devuelva el número más pequeño.
Explicación: La función utiliza `min()` para encontrar el valor mínimo en la lista.

```
def minimo_lista(lista):
    return min(lista)

print(minimo_lista([5, 7, 2, 9]))
```

Ejercicio 49

Enunciado: Crea una función que reciba una lista de números y devuelva solo los números impares.
Explicación: La función filtra elementos usando una comprensión de listas.

```
def filtrar_impares(lista):
    return [x for x in lista if x % 2 != 0]

print(filtrar_impares([1, 2, 3, 4, 5]))
```

Ejercicio 50

Enunciado: Crea una función que reciba un texto y devuelva cuántas vocales contiene.
Explicación: La función recorre el texto y cuenta las vocales encontradas.

```
def contar_vocales(texto):
    vocales = "aeiouAEIOU"
    return sum(1 for letra in texto if letra in vocales)

print(contar_vocales("Hola Mundo"))
```

Ejercicio 51

Enunciado: Crea una función que reciba un número y devuelva su raíz cúbica.
Explicación: La función realiza la operación matemática usando `** (1/3)`.

```
def raiz_cubica(n):
    return n ** (1/3)


print(raiz_cubica(27))
```

Ejercicio 52

Enunciado: Crea una función que reciba una lista y devuelva el número de elementos pares.
Explicación: La función recorre la lista y cuenta los elementos que cumplen la condición de ser pares.

```
def contar_pares(lista):
    return sum(1 for x in lista if x % 2 == 0)


print(contar_pares([1, 2, 3, 4, 5, 6]))
```

Ejercicio 53

Enunciado: Crea una función que reciba un número y devuelva su inverso (1/n).
Explicación: La función devuelve el inverso, cuidando que el número no sea cero.

```
def inverso(n):
    if n != 0:
        return 1 / n
    else:
        return "Error: división por cero"

print(inverso(5))
print(inverso(0))
```

Ejercicio 54

Enunciado: Crea una función que reciba un número y devuelva True si es múltiplo de 7.
Explicación: Evalúa la divisibilidad usando % y devuelve un booleano.

```
def multiplo_siete(n):
    return n % 7 == 0

print(multiplo_siete(14))
print(multiplo_siete(10))
```

Ejercicio 55

Enunciado: Crea una función que reciba dos números y devuelva su promedio.
Explicación: La función realiza la operación matemática y devuelve el resultado.

```
def promedio_dos(a, b):
    return (a + b) / 2

print(promedio_dos(8, 12))
```

Ejercicio 56

Enunciado: Crea una función que reciba una lista y devuelva la lista sin elementos repetidos.
Explicación: La función usa `set()` para eliminar duplicados y `list()` para convertir de nuevo a lista.

```
def eliminar_duplicados(lista):
    return list(set(lista))

print(eliminar_duplicados([1, 2, 2, 3, 3, 4]))
```

Ejercicio 57

Enunciado: Crea una función que reciba un número y devuelva True si es impar.
Explicación: Evalúa la paridad usando `%` y devuelve un valor booleano.

```
def es_impar(n):
    return n % 2 != 0

print(es_impar(7))
print(es_impar(4))
```

Ejercicio 58

Enunciado: Crea una función que reciba una cadena de texto y devuelva la misma sin espacios.
Explicación: La función utiliza `replace()` para eliminar los espacios.

```
def quitar_espacios(texto):
    return texto.replace(" ", "")

print(quitar_espacios("Hola Mundo"))
```

Ejercicio 59

Enunciado: Crea una función que reciba una lista de números y devuelva la lista de cuadrados.
Explicación: La función recorre la lista y calcula el cuadrado de cada elemento usando comprensión de listas.

```
def cuadrados_lista(lista):
    return [x ** 2 for x in lista]

print(cuadrados_lista([1, 2, 3, 4]))
```

Ejercicio 60

Enunciado: Crea una función que reciba un texto y devuelva el número de palabras.
Explicación: La función utiliza `split()` para separar palabras y `len()` para contarlas.

```
def contar_palabras(texto):
    return len(texto.split())

print(contar_palabras("Hola mundo desde Python"))
```

Ejercicio 61

Enunciado: Crea una función que reciba dos números y devuelva True si el primero es mayor que el segundo.
Explicación: La función evalúa la comparación y devuelve un valor booleano.

```
def mayor_que(a, b):
    return a > b

print(mayor_que(10, 5))
print(mayor_que(3, 7))
```

Ejercicio 62

Enunciado: Crea una función que reciba un número y devuelva su triple más 2.
Explicación: La función realiza operaciones matemáticas combinadas.

```
def triple_mas_dos(n):
    return n * 3 + 2

print(triple_mas_dos(4))
```

Ejercicio 63

Enunciado: Crea una función que reciba una lista y devuelva True si está vacía.
Explicación: La función evalúa la longitud de la lista o su valor booleano.

```
def lista_vacia(lista):
    return len(lista) == 0


print(lista_vacia([]))
print(lista_vacia([1, 2]))
```

Ejercicio 64

Enunciado: Crea una función que reciba un texto y devuelva True si contiene la letra “a”.
Explicación: La función utiliza el operador `in` para verificar la presencia de un carácter.

```
def contiene_a(texto):
    return 'a' in texto.lower()


print(contiene_a("Hola"))
print(contiene_a("Mundo"))
```

Ejercicio 65

Enunciado: Crea una función que reciba un número y devuelva su valor negativo.
Explicación: La función cambia el signo del número usando `-`.

```
def negativo(n):
    return -n


print(negativo(5))
print(negativo(-8))
```

Ejercicio 66

Enunciado: Crea una función que reciba una lista y devuelva la suma de sus elementos impares.
Explicación: La función recorre la lista y filtra los números impares antes de sumarlos.

```
def suma_impares(lista):
    return sum(x for x in lista if x % 2 != 0)

print(suma_impares([1, 2, 3, 4, 5]))
```

Ejercicio 67

Enunciado: Crea una función que reciba un número y devuelva True si es divisible por 2 o 5.
Explicación: Evalúa dos condiciones usando `or` y devuelve un valor booleano.

```
def divisible_dos_o_cinco(n):
    return n % 2 == 0 or n % 5 == 0

print(divisible_dos_o_cinco(10))
print(divisible_dos_o_cinco(7))
```

Ejercicio 68

Enunciado: Crea una función que reciba una lista y devuelva True si todos sus elementos son positivos.
Explicación: La función utiliza un bucle o `all()` para evaluar cada elemento.

```
def todos_positivos(lista):
    return all(x > 0 for x in lista)

print(todos_positivos([1, 2, 3]))
print(todos_positivos([1, -2, 3]))
```

Ejercicio 69

Enunciado: Crea una función que reciba un número y devuelva su doble si es par, o su triple si es impar.
Explicación: La función usa condicionales para decidir qué operación realizar.

```
def doble_o_triple(n):
    if n % 2 == 0:
        return n * 2
    else:
        return n * 3

print(doble_o_triple(4))
print(doble_o_triple(5))
```

Ejercicio 70

Enunciado: Crea una función que reciba un texto y devuelva el número de letras “o” que contiene.
Explicación: La función recorre el texto y cuenta las ocurrencias de un carácter específico.

```
def contar_o(texto):
    return sum(1 for letra in texto.lower() if letra == 'o')

print(contar_o("Hola Mundo"))
```

Ejercicio 71

Enunciado: Crea una función que reciba una lista y devuelva True si contiene el número 0.
Explicación: La función utiliza el operador `in` para verificar la presencia de un elemento en la lista.

```
def contiene_cero(lista):
    return 0 in lista

print(contiene_cero([1, 2, 3, 0]))
print(contiene_cero([1, 2, 3]))
```

Ejercicio 72

Enunciado: Crea una función que reciba un número y devuelva True si es divisible por 10.
Explicación: La función evalúa la divisibilidad usando `%` y devuelve un valor booleano.

```
def divisible_diez(n):
    return n % 10 == 0

print(divisible_diez(50))
print(divisible_diez(23))
```

Ejercicio 73

Enunciado: Crea una función que reciba un texto y devuelva la primera letra.
Explicación: La función accede al primer carácter de la cadena usando índice 0.

```
def primera_letra(texto):
    return texto[0] if texto else None

print(primera_letra("Python"))
print(primera_letra(""))
```

Ejercicio 74

Enunciado: Crea una función que reciba una lista y devuelva True si todos los elementos son pares.
Explicación: La función usa `all()` y una expresión generadora para evaluar la condición en todos los elementos.

```
def todos_pares(lista):
    return all(x % 2 == 0 for x in lista)

print(todos_pares([2, 4, 6]))
print(todos_pares([2, 3, 4]))
```

Ejercicio 75

Enunciado: Crea una función que reciba dos números y devuelva la suma si son distintos, o cero si son iguales.
Explicación: La función usa condicionales para determinar qué valor devolver.

```
def suma_si_distintos(a, b):
    if a != b:
        return a + b
    else:
        return 0

print(suma_si_distintos(5, 3))
print(suma_si_distintos(4, 4))
```

Ejercicio 76

Enunciado: Crea una función que reciba un número y devuelva True si es múltiplo de 3 o 7.
Explicación: Evalúa dos condiciones usando `or` y devuelve un valor booleano.

```
def multiplo_tres_siete(n):
    return n % 3 == 0 or n % 7 == 0

print(multiplo_tres_siete(9))
print(multiplo_tres_siete(14))
print(multiplo_tres_siete(10))
```

Ejercicio 77

Enunciado: Crea una función que reciba una lista y devuelva la lista con sus elementos al cuadrado.
Explicación: La función utiliza comprensión de listas para procesar cada elemento.

```
def lista_cuadrados(lista):
    return [x ** 2 for x in lista]

print(lista_cuadrados([1, 2, 3, 4]))
```

Ejercicio 78

Enunciado: Crea una función que reciba un número y devuelva True si es mayor que 0 y menor que 100.
Explicación: La función evalúa un rango usando operadores de comparación combinados con `and`.

```
def entre_0_y_100(n):
    return 0 < n < 100


print(entre_0_y_100(50))
print(entre_0_y_100(150))
```

Ejercicio 79

Enunciado: Crea una función que reciba un texto y devuelva True si empieza con “A”.
Explicación: La función usa el método `startswith()` para verificar el primer carácter.

```
def empieza_con_a(texto):
    return texto.startswith("A")


print(empieza_con_a("Ana"))
print(empieza_con_a("Pedro"))
```

Ejercicio 80

Enunciado: Crea una función que reciba una lista y devuelva True si contiene al menos un número negativo.
Explicación: La función usa `any()` y una expresión generadora para evaluar la condición en la lista.

```
def contiene_negativos(lista):
    return any(x < 0 for x in lista)


print(contiene_negativos([1, -2, 3]))
print(contiene_negativos([1, 2, 3]))
```

Ejercicio 81

Enunciado: Crea una función que reciba un número y devuelva True si es múltiplo de 2 y 5.
Explicación: La función evalúa dos condiciones simultáneamente usando `and` y devuelve un booleano.

```
def multiplo_dos_y_cinco(n):
    return n % 2 == 0 and n % 5 == 0


print(multiplo_dos_y_cinco(10))
print(multiplo_dos_y_cinco(8))
```

Ejercicio 82

Enunciado: Crea una función que reciba una lista y devuelva el segundo elemento.
Explicación: La función accede a un elemento específico usando índice 1 y controla que la lista tenga suficientes elementos.

```
def segundo_elemento(lista):
    return lista[1] if len(lista) > 1 else None


print(segundo_elemento([5, 7, 9]))
print(segundo_elemento([3]))
```

Ejercicio 83

Enunciado: Crea una función que reciba un texto y devuelva True si tiene más de 5 caracteres.
Explicación: La función evalúa la longitud de la cadena con `len()`.

```
def mas_de_cinco(texto):
    return len(texto) > 5


print(mas_de_cinco("Python"))
print(mas_de_cinco("Hola"))
```

Ejercicio 84

Enunciado: Crea una función que reciba un número y devuelva su módulo 10.
Explicación: La función utiliza el operador `%` para calcular el residuo de dividir entre 10.

```
def modulo_diez(n):
    return n % 10


print(modulo_diez(27))
print(modulo_diez(50))
```

Ejercicio 85

Enunciado: Crea una función que reciba una lista y devuelva True si todos los elementos son positivos y pares.
Explicación: La función evalúa múltiples condiciones usando `all()` y operadores lógicos.

```
def positivos_y_pares(lista):
    return all(x > 0 and x % 2 == 0 for x in lista)


print(positivos_y_pares([2, 4, 6]))
print(positivos_y_pares([2, -4, 6]))
```

Ejercicio 86

Enunciado: Crea una función que reciba dos números y devuelva True si son múltiplos uno del otro.
Explicación: La función evalúa la divisibilidad en ambos sentidos usando `%`.

```
def multiplos(a, b):
    return a % b == 0 or b % a == 0


print(multiplos(10, 5))
print(multiplos(7, 3))
```

Ejercicio 87

Enunciado: Crea una función que reciba una lista y devuelva el promedio de sus elementos.
Explicación: La función suma los elementos y divide entre la cantidad usando `len()`.

```
def promedio_lista(lista):
    return sum(lista) / len(lista) if lista else 0


print(promedio_lista([2, 4, 6, 8]))
```

Ejercicio 88

Enunciado: Crea una función que reciba un número y devuelva True si es un número negativo.
Explicación: Evalúa si el número es menor que cero y devuelve un booleano.

```
def es_negativo(n):
    return n < 0


print(es_negativo(-5))
print(es_negativo(3))
```

Ejercicio 89

Enunciado: Crea una función que reciba una lista y devuelva True si contiene al menos un número par.
Explicación: La función usa `any()` y una expresión generadora para evaluar la condición en la lista.

```
def contiene_par(lista):
    return any(x % 2 == 0 for x in lista)


print(contiene_par([1, 3, 5, 6]))
print(contiene_par([1, 3, 5]))
```

Ejercicio 90

Enunciado: Crea una función que reciba un texto y devuelva True si contiene tanto la letra “a” como la letra “e”.
Explicación: La función evalúa la presencia de dos caracteres usando `in` y operadores lógicos.

```
def contiene_a_y_e(texto):
    texto = texto.lower()
    return 'a' in texto and 'e' in texto


print(contiene_a_y_e("Pequeña"))
print(contiene_a_y_e("Hola Mundo"))
```

Ejercicio 91

Enunciado: Crea una función que reciba un número y devuelva True si es múltiplo de 3 y no múltiplo de 2.
Explicación: La función evalúa condiciones combinadas usando `and` y `not`.

```
def multiplo_tres_no_dos(n):
    return n % 3 == 0 and n % 2 != 0


print(multiplo_tres_no_dos(9))
print(multiplo_tres_no_dos(6))
```

Ejercicio 92

Enunciado: Crea una función que reciba una lista y devuelva el penúltimo elemento.
Explicación: La función accede a elementos usando índices negativos para encontrar el penúltimo.

```
def penultimo_elemento(lista):
    return lista[-2] if len(lista) > 1 else None


print(penultimo_elemento([1, 2, 3, 4]))
print(penultimo_elemento([5]))
```

Ejercicio 93

Enunciado: Crea una función que reciba un texto y devuelva True si empieza y termina con la misma letra.
Explicación: La función compara el primer y último carácter usando índices.

```
def empieza_termina_igual(texto):
    return texto[0].lower() == texto[-1].lower() if texto else False


print(empieza_termina_igual("radar"))
print(empieza_termina_igual("Python"))
```

Ejercicio 94

Enunciado: Crea una función que reciba dos números y devuelva su diferencia absoluta.
Explicación: La función usa `abs()` para garantizar un resultado positivo.

```
def diferencia_absoluta(a, b):
    return abs(a - b)


print(diferencia_absoluta(5, 10))
print(diferencia_absoluta(12, 7))
```

Ejercicio 95

Enunciado: Crea una función que reciba una lista y devuelva True si contiene al menos un número negativo.
Explicación: La función usa `any()` y una expresión generadora para detectar valores negativos.

```
def hay_negativos(lista):
    return any(x < 0 for x in lista)


print(hay_negativos([1, -2, 3]))
print(hay_negativos([1, 2, 3]))
```

Ejercicio 96

Enunciado: Crea una función que reciba un número y devuelva True si está entre 50 y 100 (inclusive).
Explicación: La función evalúa un rango usando operadores de comparación.

```
def entre_50_y_100(n):
    return 50 <= n <= 100


print(entre_50_y_100(75))
print(entre_50_y_100(120))
```

Ejercicio 97

Enunciado: Crea una función que reciba una lista y devuelva la suma de sus elementos positivos.
Explicación: La función recorre la lista y suma solo los valores mayores que cero.

```
def suma_positivos(lista):
    return sum(x for x in lista if x > 0)


print(suma_positivos([1, -2, 3, -4, 5]))
```

Ejercicio 98

Enunciado: Crea una función que reciba un texto y devuelva True si contiene solo letras.
Explicación: La función utiliza `isalpha()` para verificar que todos los caracteres sean letras.

```
def solo_letras(texto):
    return texto.isalpha()


print(solo_letras("Python"))
print(solo_letras("Python3"))
```

Ejercicio 99

Enunciado: Crea una función que reciba dos números y devuelva el mayor.
Explicación: La función compara los dos valores y devuelve el mayor usando condicionales o `max()`.

```
def mayor_numero(a, b):
    return max(a, b)


print(mayor_numero(10, 20))
```

Ejercicio 100

Enunciado: Crea una función que reciba un texto y devuelva la cantidad de letras mayúsculas que contiene.
Explicación: La función recorre el texto y cuenta los caracteres que son mayúsculas usando `isupper()`.

```
def contar_mayusculas(texto):
    return sum(1 for letra in texto if letra.isupper())


print(contar_mayusculas("Hola Mundo Python"))
```

Ejercicios python programacion orientada a objetos

Ejercicio 1

Enunciado: Crea una clase `Persona` con atributos `nombre` y `edad`, y un método que imprima un saludo.
Explicación: Se introduce la creación de clases, atributos y métodos.

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def saludar(self):
        print(f"Hola, me llamo {self.nombre} y tengo {self.edad}
años.")

persona1 = Persona("Ana", 25)
persona1.saludar()
```

Ejercicio 2

Enunciado: Crea una clase `Rectangulo` con atributos `base` y `altura` y un método que calcule el área.
Explicación: Introduce métodos que realizan cálculos usando atributos de la clase.

```
class Rectangulo:
    def __init__(self, base, altura):
        self.base = base
        self.altura = altura

    def area(self):
        return self.base * self.altura

r = Rectangulo(5, 3)
print("Área:", r.area())
```

Ejercicio 3

Enunciado: Crea una clase `Coche` con atributos `marca` y `modelo` y un método que imprima la descripción del coche.
Explicación: Practica la creación de objetos y métodos que acceden a sus atributos.

```
class Coche:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def descripcion(self):
        print(f"Coche: {self.marca} {self.modelo}")

mi_coche = Coche("Toyota", "Corolla")
mi_coche.descripcion()
```

Ejercicio 4

Enunciado: Crea una clase `CuentaBancaria` con atributos `titular` y `saldo`, y métodos para depositar y retirar dinero.
Explicación: Introduce métodos que modifican los atributos de la clase.

```
class CuentaBancaria:
    def __init__(self, titular, saldo):
        self.titular = titular
        self.saldo = saldo

    def depositar(self, cantidad):
        self.saldo += cantidad

    def retirar(self, cantidad):
        if cantidad <= self.saldo:
            self.saldo -= cantidad
        else:
            print("Saldo insuficiente")

cuenta = CuentaBancaria("Luis", 1000)
cuenta.depositar(500)
cuenta.retirar(200)
print("Saldo:", cuenta.saldo)
```

Ejercicio 5

Enunciado: Crea una clase `Animal` con un atributo `nombre` y un método `hablar()` que imprima un sonido genérico.
Explicación: Se introduce la idea de métodos que representan acciones de un objeto.

```
class Animal:
    def __init__(self, nombre):
        self.nombre = nombre

    def hablar(self):
        print(f"{self.nombre} hace un sonido")

animal = Animal("Perro")
animal.hablar()
```

Ejercicio 6

Enunciado: Crea una clase `Libro` con atributos `titulo` y `autor`, y un método que imprima la información del libro.
Explicación: Práctica de clases sencillas con métodos que muestran datos.

```
class Libro:
    def __init__(self, titulo, autor):
        self.titulo = titulo
        self.autor = autor

    def info(self):
        print(f"'{self.titulo}' escrito por {self.autor}")

libro = Libro("1984", "George Orwell")
libro.info()
```

Ejercicio 7

Enunciado: Crea una clase `Empleado` con atributos `nombre` y `sueldo`, y un método que aumente el sueldo en un porcentaje.
Explicación: Introduce modificación de atributos mediante métodos.

```
class Empleado:
    def __init__(self, nombre, sueldo):
        self.nombre = nombre
        self.sueldo = sueldo

    def aumentar_sueldo(self, porcentaje):
        self.sueldo += self.sueldo * porcentaje / 100

empleado = Empleado("Carlos", 2000)
empleado.aumentar_sueldo(10)
print("Sueldo actualizado:", empleado.sueldo)
```

Ejercicio 8

Enunciado: Crea una clase `Estudiante` con atributos `nombre` y `notas` (lista), y un método que calcule el promedio.
Explicación: Practica manejo de listas como atributos de la clase.

```
class Estudiante:
    def __init__(self, nombre, notas):
        self.nombre = nombre
        self.notas = notas

    def promedio(self):
        return sum(self.notas) / len(self.notas)

estudiante = Estudiante("Lucía", [8, 9, 7])
print("Promedio:", estudiante.promedio())
```

Ejercicio 9

Enunciado: Crea una clase `Producto` con atributos `nombre` y `precio`, y un método que aplique un descuento.
Explicación: Introduce métodos que reciben parámetros y modifican atributos.

```
class Producto:
    def __init__(self, nombre, precio):
        self.nombre = nombre
        self.precio = precio

    def aplicar_descuento(self, porcentaje):
        self.precio -= self.precio * porcentaje / 100

producto = Producto("Laptop", 1000)
producto.aplicar_descuento(10)
print("Precio con descuento:", producto.precio)
```

Ejercicio 10

Enunciado: Crea una clase `Mascota` con atributos `nombre` y `tipo`, y un método que describa la mascota.
Explicación: Refuerza la creación de clases y métodos para mostrar información del objeto.

```
class Mascota:
    def __init__(self, nombre, tipo):
        self.nombre = nombre
        self.tipo = tipo

    def descripcion(self):
        print(f"Tengo un {self.tipo} llamado {self.nombre}")

mascota = Mascota("Fido", "perro")
mascota.descripcion()
```

Ejercicio 11

Enunciado: Crea una clase `Perro` que herede de `Animal` y sobreescriba el método `hablar()` para que diga “Guau”.
Explicación: Introduce herencia y sobrescritura de métodos.

```
class Animal:
    def __init__(self, nombre):
        self.nombre = nombre

    def hablar(self):
        print(f"{self.nombre} hace un sonido")

class Perro(Animal):
    def hablar(self):
        print(f"{self.nombre} dice Guau")

mi_perro = Perro("Firulais")
mi_perro.hablar()
```

Ejercicio 12

Enunciado: Crea una clase `Gato` que herede de `Animal` y sobreescriba el método `hablar()` para que diga “Miau”.
Explicación: Practica la herencia y personalización de métodos.

```
class Gato(Animal):
    def hablar(self):
        print(f"{self.nombre} dice Miau")

mi_gato = Gato("Mishi")
mi_gato.hablar()
```

Ejercicio 13

Enunciado: Crea una clase `CuentaBancariaSegura` que herede de `CuentaBancaria` y agregue un método `mostrar_saldo()` que lo imprima.
Explicación: Introduce herencia y métodos adicionales en la subclase.

```
class CuentaBancaria:
    def __init__(self, titular, saldo):
        self.titular = titular
        self.saldo = saldo

    def depositar(self, cantidad):
        self.saldo += cantidad

class CuentaBancariaSegura(CuentaBancaria):
    def mostrar_saldo(self):
        print(f"Saldo: {self.saldo}")

cuenta = CuentaBancariaSegura("Ana", 1000)
cuenta.depositar(500)
cuenta.mostrar_saldo()
```

Ejercicio 14

Enunciado: Crea una clase `Triangulo` con atributos `base` y `altura` y un método que calcule el área.
Explicación: Similar a rectángulo, pero con la fórmula de triángulo.

```
class Triangulo:
    def __init__(self, base, altura):
        self.base = base
        self.altura = altura

    def area(self):
        return (self.base * self.altura) / 2

t = Triangulo(4, 6)
print("Área del triángulo:", t.area())
```

Ejercicio 15

Enunciado: Crea una clase `PersonaConEmail` que herede de `Persona` y agregue un atributo `email`.
Explicación: Practica herencia y ampliación de atributos de la clase base.

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

class PersonaConEmail(Persona):
    def __init__(self, nombre, edad, email):
        super().__init__(nombre, edad)
        self.email = email

p = PersonaConEmail("Luis", 30, "luis@example.com")
print(p.nombre, p.edad, p.email)
```

Ejercicio 16

Enunciado: Crea una clase `Vehiculo` con atributos `marca` y `modelo` y un método `detalle()` que los imprima.
Explicación: Introduce clases genéricas de objetos cotidianos.

```
class Vehiculo:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def detalle(self):
        print(f"Vehículo: {self.marca} {self.modelo}")

v = Vehiculo("Ford", "Focus")
v.detalle()
```

Ejercicio 17

Enunciado: Crea una clase `Auto` que herede de `Vehiculo` y agregue un atributo `color`.
Explicación: Practica herencia y extensión de atributos.

```
class Auto(Vehiculo):
    def __init__(self, marca, modelo, color):
        super().__init__(marca, modelo)
        self.color = color

auto = Auto("Toyota", "Corolla", "Rojo")
print(auto.marca, auto.modelo, auto.color)
```

Ejercicio 18

Enunciado: Crea una clase `Alumno` con atributos `nombre` y `notas` y un método que diga si aprobó (promedio ≥ 6).
Explicación: Introduce métodos que devuelven valores booleanos según atributos.

```
class Alumno:
    def __init__(self, nombre, notas):
        self.nombre = nombre
        self.notas = notas

    def aprobado(self):
        return sum(self.notas)/len(self.notas) >= 6

alumno = Alumno("Sara", [7, 8, 5])
print(alumno.aprobado())
```

Ejercicio 19

Enunciado: Crea una clase `ProductoConStock` que herede de `Producto` y agregue un atributo `stock`.
Explicación: Combina herencia y nuevos atributos para la subclase.

```
class Producto:
    def __init__(self, nombre, precio):
        self.nombre = nombre
        self.precio = precio

class ProductoConStock(Producto):
    def __init__(self, nombre, precio, stock):
        super().__init__(nombre, precio)
        self.stock = stock

p = ProductoConStock("Mouse", 20, 50)
print(p.nombre, p.precio, p.stock)
```

Ejercicio 20

Enunciado: Crea una clase `MascotaConEdad` que herede de `Mascota` y agregue un atributo `edad`.
Explicación: Practica herencia y adición de atributos a la clase base.

```
class Mascota:
    def __init__(self, nombre, tipo):
        self.nombre = nombre
        self.tipo = tipo

class MascotaConEdad(Mascota):
    def __init__(self, nombre, tipo, edad):
        super().__init__(nombre, tipo)
        self.edad = edad

m = MascotaConEdad("Fido", "perro", 5)
print(m.nombre, m.tipo, m.edad)
```

Ejercicio 21

Enunciado: Crea una clase `Circulo` con atributo `radio` y un método que calcule su área.
Explicación: Practica el uso de atributos y métodos que realizan cálculos matemáticos.

```
import math

class Circulo:
    def __init__(self, radio):
        self.radio = radio

    def area(self):
        return math.pi * self.radio ** 2

c = Circulo(3)
print("Área del círculo:", c.area())
```

Ejercicio 22

Enunciado: Crea una clase `CuentaBancariaVIP` que herede de `CuentaBancaria` y tenga un método `bono()` que agregue 100 al saldo.
Explicación: Introduce herencia y métodos adicionales para la subclase.

```
class CuentaBancaria:
    def __init__(self, titular, saldo):
        self.titular = titular
        self.saldo = saldo

    def depositar(self, cantidad):
        self.saldo += cantidad

class CuentaBancariaVIP(CuentaBancaria):
    def bono(self):
        self.saldo += 100

cuenta = CuentaBancariaVIP("Ana", 1000)
cuenta.bono()
print("Saldo con bono:", cuenta.saldo)
```

Ejercicio 23

Enunciado: Crea una clase `LibroDigital` que herede de `Libro` y agregue un atributo `formato`.
Explicación: Combina herencia y nuevos atributos específicos.

```
class Libro:
    def __init__(self, titulo, autor):
        self.titulo = titulo
        self.autor = autor

class LibroDigital(Libro):
    def __init__(self, titulo, autor, formato):
        super().__init__(titulo, autor)
        self.formato = formato

ld = LibroDigital("1984", "George Orwell", "PDF")
print(ld.titulo, ld.autor, ld.formato)
```

Ejercicio 24

Enunciado: Crea una clase `CuentaCorriente` que herede de `CuentaBancaria` y agregue un atributo `limite`.
Explicación: Practica la herencia y modificación de atributos de la subclase.

```
class CuentaCorriente(CuentaBancaria):
    def __init__(self, titular, saldo, limite):
        super().__init__(titular, saldo)
        self.limite = limite

cuenta = CuentaCorriente("Luis", 1000, 500)
print(cuenta.titular, cuenta.saldo, cuenta.limite)
```

Ejercicio 25

Enunciado: Crea una clase `EmpleadoTemporal` que herede de `Empleado` y agregue un atributo `meses_contrato`.
Explicación: Introduce herencia y extensión de atributos.

```
class Empleado:
    def __init__(self, nombre, sueldo):
        self.nombre = nombre
        self.sueldo = sueldo

class EmpleadoTemporal(Empleado):
    def __init__(self, nombre, sueldo, meses_contrato):
        super().__init__(nombre, sueldo)
        self.meses_contrato = meses_contrato

empleado = EmpleadoTemporal("Ana", 1500, 6)
print(empleado.nombre, empleado.sueldo, empleado.meses_contrato)
```

Ejercicio 26

Enunciado: Crea una clase `Profesor` con atributos `nombre` y `asignatura` y un método que imprima su presentación.
Explicación: Introduce clases básicas con métodos que muestran información.

```
class Profesor:
    def __init__(self, nombre, asignatura):
        self.nombre = nombre
        self.asignatura = asignatura

    def presentarse(self):
        print(f"Soy {self.nombre} y enseño {self.asignatura}")

profesor = Profesor("Carlos", "Matemáticas")
profesor.presentarse()
```

Ejercicio 27

Enunciado: Crea una clase `EstudianteConEmail` que herede de `Estudiante` y agregue un atributo `email`.
Explicación: Practica herencia y extensión de atributos en la subclase.

```
class Estudiante:
    def __init__(self, nombre, notas):
        self.nombre = nombre
        self.notas = notas

class EstudianteConEmail(Estudiante):
    def __init__(self, nombre, notas, email):
        super().__init__(nombre, notas)
        self.email = email

est = EstudianteConEmail("Lucía", [8, 7, 9], "lucia@example.com")
print(est.nombre, est.notas, est.email)
```

Ejercicio 28

Enunciado: Crea una clase `Bicicleta` con atributos `marca` y `tipo` y un método que imprima sus características.
Explicación: Introduce clases y métodos que muestran los atributos.

```
class Bicicleta:
    def __init__(self, marca, tipo):
        self.marca = marca
        self.tipo = tipo

    def descripcion(self):
        print(f"Bicicleta {self.marca}, tipo {self.tipo}")

bici = Bicicleta("Trek", "Montaña")
bici.descripcion()
```

Ejercicio 29

Enunciado: Crea una clase `VehiculoElectrico` que herede de `Vehiculo` y agregue un atributo `bateria`.
Explicación: Practica herencia y adición de atributos específicos a la subclase.

```
class Vehiculo:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

class VehiculoElectrico(Vehiculo):
    def __init__(self, marca, modelo, bateria):
        super().__init__(marca, modelo)
        self.bateria = bateria

ve = VehiculoElectrico("Tesla", "Model 3", 75)
print(ve.marca, ve.modelo, ve.bateria)
```

Ejercicio 30

Enunciado: Crea una clase `Smartphone` con atributos `marca`, `modelo` y un método que imprima su descripción.
Explicación: Introduce clases y métodos básicos para objetos tecnológicos.

```
class Smartphone:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def descripcion(self):
        print(f"Smartphone: {self.marca} {self.modelo}")

telefono = Smartphone("Samsung", "Galaxy S21")
telefono.descripcion()
```

Ejercicio 31

Enunciado: Crea una clase `TrianguloRectangulo` que herede de `Triangulo` y agregue un método que calcule la hipotenusa.
Explicación: Introduce herencia y cálculos adicionales en la subclase.

```
import math

class Triangulo:
    def __init__(self, base, altura):
        self.base = base
        self.altura = altura

    def area(self):
        return (self.base * self.altura) / 2

class TrianguloRectangulo(Triangulo):
    def hipotenusa(self):
        return math.sqrt(self.base**2 + self.altura**2)

t = TrianguloRectangulo(3, 4)
print("Área:", t.area())
print("Hipotenusa:", t.hipotenusa())
```

Ejercicio 32

Enunciado: Crea una clase `PersonaConTelefono` que herede de `Persona` y agregue un atributo `telefono`.
Explicación: Practica herencia y extensión de atributos.

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

class PersonaConTelefono(Persona):
    def __init__(self, nombre, edad, telefono):
        super().__init__(nombre, edad)
        self.telefono = telefono

p = PersonaConTelefono("Ana", 28, "123456789")
print(p.nombre, p.edad, p.telefono)
```

Ejercicio 33

Enunciado: Crea una clase `CuentaConInteres` que herede de `CuentaBancaria` y agregue un método `aplicar_interes()` que incremente el saldo un 5%.
Explicación: Introduce métodos que modifican atributos basados en reglas.

```
class CuentaBancaria:
    def __init__(self, titular, saldo):
        self.titular = titular
        self.saldo = saldo

class CuentaConInteres(CuentaBancaria):
    def aplicar_interes(self):
        self.saldo += self.saldo * 0.05

cuenta = CuentaConInteres("Luis", 1000)
cuenta.aplicar_interes()
print("Saldo con interés:", cuenta.saldo)
```

Ejercicio 34

Enunciado: Crea una clase `LibroFisico` que herede de `Libro` y agregue un atributo `paginas`.
Explicación: Practica herencia y extensión de atributos específicos.

```
class Libro:
    def __init__(self, titulo, autor):
        self.titulo = titulo
        self.autor = autor

class LibroFisico(Libro):
    def __init__(self, titulo, autor, paginas):
        super().__init__(titulo, autor)
        self.paginas = paginas

libro = LibroFisico("El Principito", "Saint-Exupéry", 96)
print(libro.titulo, libro.autor, libro.paginas)
```

Ejercicio 35

Enunciado: Crea una clase `EmpleadoConBono` que herede de `Empleado` y agregue un método `aplicar_bono()` que aumente el sueldo 200 unidades.
Explicación: Introduce métodos adicionales que modifican atributos.

```
class Empleado:
    def __init__(self, nombre, sueldo):
        self.nombre = nombre
        self.sueldo = sueldo

class EmpleadoConBono(Empleado):
    def aplicar_bono(self):
        self.sueldo += 200

empleado = EmpleadoConBono("Carlos", 1500)
empleado.aplicar_bono()
print("Sueldo con bono:", empleado.sueldo)
```

Ejercicio 36

Enunciado: Crea una clase `EstudianteConPromocion` que herede de `Estudiante` y agregue un método que indique si el promedio es sobresaliente (>9).
Explicación: Introduce métodos que evalúan condiciones sobre atributos de la clase.

```
class Estudiante:
    def __init__(self, nombre, notas):
        self.nombre = nombre
        self.notas = notas

class EstudianteConPromocion(Estudiante):
    def sobresaliente(self):
        return sum(self.notas)/len(self.notas) > 9

est = EstudianteConPromocion("Lucía", [10, 9, 9])
print(est.sobresaliente())
```

Ejercicio 37

Enunciado: Crea una clase `SmartphoneConAlmacenamiento` que herede de `Smartphone` y agregue un atributo `almacenamiento`.
Explicación: Practica herencia y extensión de atributos en objetos tecnológicos.

```
class Smartphone:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

class SmartphoneConAlmacenamiento(Smartphone):
    def __init__(self, marca, modelo, almacenamiento):
        super().__init__(marca, modelo)
        self.almacenamiento = almacenamiento

telefono = SmartphoneConAlmacenamiento("Apple", "iPhone 14",
"128GB")
print(telefono.marca, telefono.modelo, telefono.almacenamiento)
```

Ejercicio 38

Enunciado: Crea una clase `PerroConRaza` que herede de `Perro` y agregue un atributo `raza`.
Explicación: Introduce herencia y atributos específicos de subclases.

```
class Perro:
    def __init__(self, nombre):
        self.nombre = nombre

    def hablar(self):
        print(f"{self.nombre} dice Guau")

class PerroConRaza(Perro):
    def __init__(self, nombre, raza):
        super().__init__(nombre)
        self.raza = raza

perro = PerroConRaza("Firulais", "Labrador")
print(perro.nombre, perro.raza)
perro.hablar()
```

Ejercicio 39

Enunciado: Crea una clase `AlumnoConEdad` que herede de `Alumno` y agregue un atributo `edad`.
Explicación: Combina herencia con nuevos atributos en la subclase.

```
class Alumno:
    def __init__(self, nombre, notas):
        self.nombre = nombre
        self.notas = notas

class AlumnoConEdad(Alumno):
    def __init__(self, nombre, notas, edad):
        super().__init__(nombre, notas)
        self.edad = edad

alumno = AlumnoConEdad("Sara", [7, 8, 6], 17)
print(alumno.nombre, alumno.notas, alumno.edad)
```

Ejercicio 40

Enunciado: Crea una clase `BicicletaConColor` que herede de `Bicicleta` y agregue un atributo `color`.
Explicación: Introduce herencia y personalización de objetos con nuevos atributos.

```
class Bicicleta:
    def __init__(self, marca, tipo):
        self.marca = marca
        self.tipo = tipo

class BicicletaConColor(Bicicleta):
    def __init__(self, marca, tipo, color):
        super().__init__(marca, tipo)
        self.color = color

bici = BicicletaConColor("Trek", "Montaña", "Rojo")
print(bici.marca, bici.tipo, bici.color)
```

Ejercicio 41

Enunciado: Crea una clase `VehiculoConPuertas` que herede de `Vehiculo` y agregue un atributo `puertas`.
Explicación: Practica herencia y la adición de atributos a la subclase.

```
class Vehiculo:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

class VehiculoConPuertas(Vehiculo):
    def __init__(self, marca, modelo, puertas):
        super().__init__(marca, modelo)
        self.puertas = puertas

v = VehiculoConPuertas("Honda", "Civic", 4)
print(v.marca, v.modelo, v.puertas)
```

Ejercicio 42

Enunciado: Crea una clase `PerroConEdad` que herede de `Perro` y agregue un atributo `edad`.
Explicación: Practica herencia y extensión de atributos para subclases de animales.

```
class Perro:
    def __init__(self, nombre):
        self.nombre = nombre

    def hablar(self):
        print(f"{self.nombre} dice Guau")

class PerroConEdad(Perro):
    def __init__(self, nombre, edad):
        super().__init__(nombre)
        self.edad = edad

perro = PerroConEdad("Max", 5)
print(perro.nombre, perro.edad)
perro.hablar()
```

Ejercicio 43

Enunciado: Crea una clase `EmpleadoConDepartamento` que herede de `Empleado` y agregue un atributo `departamento`.
Explicación: Practica herencia y extensión de atributos en subclases de personas.

```
class Empleado:
    def __init__(self, nombre, sueldo):
        self.nombre = nombre
        self.sueldo = sueldo

class EmpleadoConDepartamento(Empleado):
    def __init__(self, nombre, sueldo, departamento):
        super().__init__(nombre, sueldo)
        self.departamento = departamento

empleado = EmpleadoConDepartamento("Ana", 2000, "Ventas")
print(empleado.nombre, empleado.sueldo, empleado.departamento)
```

Ejercicio 44

Enunciado: Crea una clase `EstudianteConCurso` que herede de `Estudiante` y agregue un atributo `curso`.
Explicación: Combina herencia con nuevos atributos en subclases educativas.

```
class Estudiante:
    def __init__(self, nombre, notas):
        self.nombre = nombre
        self.notas = notas

class EstudianteConCurso(Estudiante):
    def __init__(self, nombre, notas, curso):
        super().__init__(nombre, notas)
        self.curso = curso

estudiante = EstudianteConCurso("Luis", [7,8,9], "Matemáticas")
print(estudiante.nombre, estudiante.notas, estudiante.curso)
```

Ejercicio 45

Enunciado: Crea una clase `SmartphoneConCamara` que herede de `Smartphone` y agregue un atributo `megapixeles`.
Explicación: Introduce herencia y atributos específicos de tecnología.

```
class Smartphone:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

class SmartphoneConCamara(Smartphone):
    def __init__(self, marca, modelo, megapixeles):
        super().__init__(marca, modelo)
        self.megapixeles = megapixeles

telefono = SmartphoneConCamara("Samsung", "S21", 108)
print(telefono.marca, telefono.modelo, telefono.megapixeles)
```

Ejercicio 46

Enunciado: Crea una clase `BicicletaConMarchas` que herede de `Bicicleta` y agregue un atributo `numero_marchas`.
Explicación: Practica herencia y adición de atributos de subclase.

```
class Bicicleta:
    def __init__(self, marca, tipo):
        self.marca = marca
        self.tipo = tipo

class BicicletaConMarchas(Bicicleta):
    def __init__(self, marca, tipo, numero_marchas):
        super().__init__(marca, tipo)
        self.numero_marchas = numero_marchas

bici = BicicletaConMarchas("Specialized", "Ruta", 21)
print(bici.marca, bici.tipo, bici.numero_marchas)
```

Ejercicio 47

Enunciado: Crea una clase `CuentaConRetiroMinimo` que herede de `CuentaBancaria` y agregue un límite mínimo de retiro.
Explicación: Introduce métodos que validan condiciones antes de modificar atributos.

```
class CuentaBancaria:
    def __init__(self, titular, saldo):
        self.titular = titular
        self.saldo = saldo

class CuentaConRetiroMinimo(CuentaBancaria):
    def retirar(self, cantidad):
        if cantidad <= self.saldo and cantidad <= 500:
            self.saldo -= cantidad
        else:
            print("Retiro no permitido")

cuenta = CuentaConRetiroMinimo("Luis", 1000)
cuenta.retirar(400)
print(cuenta.saldo)
cuenta.retirar(600)
```

Ejercicio 48

Enunciado: Crea una clase `ProfesorConEmail` que herede de `Profesor` y agregue un atributo `email`.
Explicación: Practica herencia y extensión de atributos de la subclase.

```
class Profesor:
    def __init__(self, nombre, asignatura):
        self.nombre = nombre
        self.asignatura = asignatura

class ProfesorConEmail(Profesor):
    def __init__(self, nombre, asignatura, email):
        super().__init__(nombre, asignatura)
        self.email = email

profesor = ProfesorConEmail("Carlos", "Historia",
"carlos@escuela.com")
print(profesor.nombre, profesor.asignatura, profesor.email)
```

Ejercicio 49

Enunciado: Crea una clase `AlumnoConPromocion` que herede de `Alumno` y agregue un método `promocion()` que diga si el promedio es ≥7.
Explicación: Introduce métodos que evalúan condiciones basadas en atributos heredados.

```
class Alumno:
    def __init__(self, nombre, notas):
        self.nombre = nombre
        self.notas = notas

class AlumnoConPromocion(Alumno):
    def promocion(self):
        return sum(self.notas)/len(self.notas) >= 7

alumno = AlumnoConPromocion("Sara", [6,8,7])
print(alumno.promocion())
```

Ejercicio 50

Enunciado: Crea una clase `MascotaConColor` que herede de `Mascota` y agregue un atributo `color`.
Explicación: Practica herencia y personalización de objetos con nuevos atributos.

```
class Mascota:
    def __init__(self, nombre, tipo):
        self.nombre = nombre
        self.tipo = tipo

class MascotaConColor(Mascota):
    def __init__(self, nombre, tipo, color):
        super().__init__(nombre, tipo)
        self.color = color

mascota = MascotaConColor("Fido", "Perro", "Marrón")
print(mascota.nombre, mascota.tipo, mascota.color)
```

Ejercicio 51

Enunciado: Crea una clase `CuentaAhorro` que herede de `CuentaBancaria` y agregue un método `interes_anual()` que calcule un 3% de interés sobre el saldo.
Explicación: Introduce métodos que realizan cálculos financieros simples sobre atributos heredados.

```
class CuentaBancaria:
    def __init__(self, titular, saldo):
        self.titular = titular
        self.saldo = saldo

class CuentaAhorro(CuentaBancaria):
    def interes_anual(self):
        return self.saldo * 0.03

cuenta = CuentaAhorro("Ana", 1000)
print("Interés anual:", cuenta.interes_anual())
```

Ejercicio 52

Enunciado: Crea una clase `EmpleadoConAntiguedad` que herede de `Empleado` y agregue un atributo `años_trabajados`.
Explicación: Practica herencia y adición de atributos específicos para subclases.

```
class Empleado:
    def __init__(self, nombre, sueldo):
        self.nombre = nombre
        self.sueldo = sueldo

class EmpleadoConAntiguedad(Empleado):
    def __init__(self, nombre, sueldo, años_trabajados):
        super().__init__(nombre, sueldo)
        self.años_trabajados = años_trabajados

empleado = EmpleadoConAntiguedad("Luis", 2000, 5)
print(empleado.nombre, empleado.sueldo, empleado.años_trabajados)
```

Ejercicio 53

Enunciado: Crea una clase `LibroConGenero` que herede de `Libro` y agregue un atributo `genero`.
Explicación: Practica herencia y atributos adicionales en subclases de libros.

```
class Libro:
    def __init__(self, titulo, autor):
        self.titulo = titulo
        self.autor = autor

class LibroConGenero(Libro):
    def __init__(self, titulo, autor, genero):
        super().__init__(titulo, autor)
        self.genero = genero

libro = LibroConGenero("1984", "George Orwell", "Distopía")
print(libro.titulo, libro.autor, libro.genero)
```

Ejercicio 54

Enunciado: Crea una clase `EstudianteConCursoYPromedio` que herede de `Estudiante` y agregue un método que devuelva el promedio redondeado.
Explicación: Introduce métodos que calculan valores basados en atributos de la subclase.

```
class Estudiante:
    def __init__(self, nombre, notas):
        self.nombre = nombre
        self.notas = notas

class EstudianteConCursoYPromedio(Estudiante):
    def promedio_redondeado(self):
        return round(sum(self.notas)/len(self.notas), 1)

estudiante = EstudianteConCursoYPromedio("Lucía", [8, 9, 7])
print("Promedio redondeado:", estudiante.promedio_redondeado())
```

Ejercicio 55

Enunciado: Crea una clase `PerroConColorYEdad` que herede de `Perro` y agregue atributos `color` y `edad`.
Explicación: Combina herencia y varios atributos adicionales.

```
class Perro:
    def __init__(self, nombre):
        self.nombre = nombre

    def hablar(self):
        print(f"{self.nombre} dice Guau")

class PerroConColorYEdad(Perro):
    def __init__(self, nombre, color, edad):
        super().__init__(nombre)
        self.color = color
        self.edad = edad

perro = PerroConColorYEdad("Max", "Negro", 4)
print(perro.nombre, perro.color, perro.edad)
perro.hablar()
```

Ejercicio 56

Enunciado: Crea una clase `SmartphoneConBateria` que herede de `Smartphone` y agregue un método que imprima el porcentaje de batería.
Explicación: Practica herencia y métodos que muestran información de atributos.

```
class Smartphone:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

class SmartphoneConBateria(Smartphone):
    def __init__(self, marca, modelo, bateria):
        super().__init__(marca, modelo)
        self.bateria = bateria

    def mostrar_bateria(self):
        print(f"Batería: {self.bateria}%")

telefono = SmartphoneConBateria("Apple", "iPhone 14", 85)
telefono.mostrar_bateria()
```

Ejercicio 57

Enunciado: Crea una clase `AlumnoConPromedioYCurso` que herede de `Alumno` y agregue un método que indique si debe hacer examen de recuperación (promedio < 6).
Explicación: Introduce métodos condicionales que evalúan atributos heredados.

```
class Alumno:
    def __init__(self, nombre, notas):
        self.nombre = nombre
        self.notas = notas

class AlumnoConPromedioYCurso(Alumno):
    def examen_recuperacion(self):
        return sum(self.notas)/len(self.notas) < 6

alumno = AlumnoConPromedioYCurso("Sara", [5,6,4])
print(alumno.examen_recuperacion())
```

Ejercicio 58

Enunciado: Crea una clase `BicicletaConVelocidad` que herede de `Bicicleta` y agregue un método que imprima la velocidad actual.
Explicación: Introduce métodos que muestran información de la subclase.

```
class Bicicleta:
    def __init__(self, marca, tipo):
        self.marca = marca
        self.tipo = tipo

class BicicletaConVelocidad(Bicicleta):
    def __init__(self, marca, tipo, velocidad):
        super().__init__(marca, tipo)
        self.velocidad = velocidad

    def mostrar_velocidad(self):
        print(f"Velocidad: {self.velocidad} km/h")

bici = BicicletaConVelocidad("Trek", "Montaña", 25)
bici.mostrar_velocidad()
```

Ejercicio 59

Enunciado: Crea una clase `CuentaConDepositoExtra` que herede de `CuentaBancaria` y agregue un método que permita un depósito extra de 50 unidades.
Explicación: Introduce métodos que incrementan atributos de manera específica.

```
class CuentaBancaria:
    def __init__(self, titular, saldo):
        self.titular = titular
        self.saldo = saldo

class CuentaConDepositoExtra(CuentaBancaria):
    def deposito_extra(self):
        self.saldo += 50

cuenta = CuentaConDepositoExtra("Ana", 1000)
cuenta.deposito_extra()
print("Saldo:", cuenta.saldo)
```

Ejercicio 60

Enunciado: Crea una clase `ProfesorConEdad` que herede de `Profesor` y agregue un atributo `edad`.
Explicación: Practica herencia y adición de atributos en subclases de personas.

```
class Profesor:
    def __init__(self, nombre, asignatura):
        self.nombre = nombre
        self.asignatura = asignatura

class ProfesorConEdad(Profesor):
    def __init__(self, nombre, asignatura, edad):
        super().__init__(nombre, asignatura)
        self.edad = edad

profesor = ProfesorConEdad("Carlos", "Matemáticas", 45)
print(profesor.nombre, profesor.asignatura, profesor.edad)
```

Ejercicio 61

Enunciado: Crea una clase `AutomovilConCombustible` que herede de `Vehiculo` y agregue un atributo `combustible` y un método que muestre su nivel.
Explicación: Practica herencia y métodos que muestran información de la subclase.

```
class Vehiculo:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

class AutomovilConCombustible(Vehiculo):
    def __init__(self, marca, modelo, combustible):
        super().__init__(marca, modelo)
        self.combustible = combustible

    def mostrar_combustible(self):
        print(f"Nivel de combustible: {self.combustible} litros")

auto = AutomovilConCombustible("Ford", "Fiesta", 40)
auto.mostrar_combustible()
```

Ejercicio 62

Enunciado: Crea una clase `AlumnoConNotasExtras` que herede de `Alumno` y agregue un método `agregar_nota` para añadir nuevas notas.
Explicación: Introduce métodos que modifican listas de atributos de la subclase.

```
class Alumno:
    def __init__(self, nombre, notas):
        self.nombre = nombre
        self.notas = notas

class AlumnoConNotasExtras(Alumno):
    def agregar_nota(self, nota):
        self.notas.append(nota)

alumno = AlumnoConNotasExtras("Sara", [6,7])
alumno.agregar_nota(9)
print(alumno.nombre, alumno.notas)
```

Ejercicio 63

Enunciado: Crea una clase `CuentaConTransferencia` que herede de `CuentaBancaria` y agregue un método `transferir` para enviar dinero a otra cuenta.
Explicación: Introduce métodos que interactúan con instancias de la misma clase.

```
class CuentaBancaria:
    def __init__(self, titular, saldo):
        self.titular = titular
        self.saldo = saldo

class CuentaConTransferencia(CuentaBancaria):
    def transferir(self, otra_cuenta, cantidad):
        if cantidad <= self.saldo:
            self.saldo -= cantidad
            otra_cuenta.saldo += cantidad

cuenta1 = CuentaConTransferencia("Ana", 1000)
cuenta2 = CuentaConTransferencia("Luis", 500)
cuenta1.transferir(cuenta2, 300)
print(cuenta1.saldo, cuenta2.saldo)
```

Ejercicio 64

Enunciado: Crea una clase `PerroConVacunas` que herede de `Perro` y agregue un atributo lista `vacunas` y un método para mostrarla.
Explicación: Introduce atributos de tipo lista y métodos que los gestionan.

```
class Perro:
    def __init__(self, nombre):
        self.nombre = nombre

    def hablar(self):
        print(f"{self.nombre} dice Guau")

class PerroConVacunas(Perro):
    def __init__(self, nombre, vacunas):
        super().__init__(nombre)
        self.vacunas = vacunas

    def mostrar_vacunas(self):
        print(f"Vacunas: {', '.join(self.vacunas)}")

perro = PerroConVacunas("Max", ["Rabia", "Moquillo"])
perro.mostrar_vacunas()
perro.hablar()
```

Ejercicio 65

Enunciado: Crea una clase `LibroConPrestamo` que herede de `Libro` y agregue un atributo `prestado` y un método `prestar`.
Explicación: Introduce métodos que cambian el estado de atributos booleanos.

```
class Libro:
    def __init__(self, titulo, autor):
        self.titulo = titulo
        self.autor = autor

class LibroConPrestamo(Libro):
    def __init__(self, titulo, autor):
        super().__init__(titulo, autor)
        self.prestado = False

    def prestar(self):
        self.prestado = True

libro = LibroConPrestamo("1984", "George Orwell")
libro.prestar()
print(libro.titulo, libro.autor, libro.prestado)
```

Ejercicio 66

Enunciado: Crea una clase `EmpleadoConAumento` que herede de `Empleado` y agregue un método `aumentar_sueldo` que aumente el sueldo un porcentaje dado.
Explicación: Introduce métodos que modifican atributos numéricos según parámetros.

```
class Empleado:
    def __init__(self, nombre, sueldo):
        self.nombre = nombre
        self.sueldo = sueldo

class EmpleadoConAumento(Empleado):
    def aumentar_sueldo(self, porcentaje):
        self.sueldo += self.sueldo * porcentaje / 100

empleado = EmpleadoConAumento("Ana", 2000)
empleado.aumentar_sueldo(10)
print(empleado.nombre, empleado.sueldo)
```

Ejercicio 67

Enunciado: Crea una clase `SmartphoneConApps` que herede de `Smartphone` y agregue un método `instalar_app` para agregar aplicaciones a una lista.
Explicación: Introduce atributos tipo lista y métodos que los gestionan.

```
class Smartphone:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

class SmartphoneConApps(Smartphone):
    def __init__(self, marca, modelo):
        super().__init__(marca, modelo)
        self.apps = []

    def instalar_app(self, app):
        self.apps.append(app)

telefono = SmartphoneConApps("Samsung", "S21")
telefono.instalar_app("WhatsApp")
telefono.instalar_app("Spotify")
print(telefono.marca, telefono.modelo, telefono.apps)
```

Ejercicio 68

Enunciado: Crea una clase `BicicletaConCambio` que herede de `Bicicleta` y agregue un método `cambiar_marcha` que modifique el atributo `numero_marchas`.
Explicación: Introduce métodos que modifican atributos de la subclase.

```
class Bicicleta:
    def __init__(self, marca, tipo):
        self.marca = marca
        self.tipo = tipo

class BicicletaConCambio(Bicicleta):
    def __init__(self, marca, tipo, numero_marchas):
        super().__init__(marca, tipo)
        self.numero_marchas = numero_marchas

    def cambiar_marcha(self, nueva_marcha):
        self.numero_marchas = nueva_marcha

bici = BicicletaConCambio("Trek", "Montaña", 21)
bici.cambiar_marcha(18)
print(bici.marca, bici.tipo, bici.numero_marchas)
```

Ejercicio 69

Enunciado: Crea una clase `AlumnoConRegistro` que herede de `Alumno` y agregue un método `agregar_registro` para añadir notas con fecha a un diccionario.
Explicación: Introduce atributos tipo diccionario y métodos para manipularlos.

```
class Alumno:
    def __init__(self, nombre):
        self.nombre = nombre

class AlumnoConRegistro(Alumno):
    def __init__(self, nombre):
        super().__init__(nombre)
        self.registro = {}

    def agregar_registro(self, fecha, nota):
        self.registro[fecha] = nota

alumno = AlumnoConRegistro("Sara")
alumno.agregar_registro("2025-10-26", 8)
print(alumno.nombre, alumno.registro)
```

Ejercicio 70

Enunciado: Crea una clase `ProfesorConAsignaturas` que herede de `Profesor` y agregue un método `agregar_asignatura` para añadir nuevas asignaturas a una lista.
Explicación: Introduce métodos para gestionar listas de atributos en subclases.

```
class Profesor:
    def __init__(self, nombre):
        self.nombre = nombre

class ProfesorConAsignaturas(Profesor):
    def __init__(self, nombre):
        super().__init__(nombre)
        self.asignaturas = []

    def agregar_asignatura(self, asignatura):
        self.asignaturas.append(asignatura)

profesor = ProfesorConAsignaturas("Carlos")
profesor.agregar_asignatura("Matemáticas")
profesor.agregar_asignatura("Física")
print(profesor.nombre, profesor.asignaturas)
```

Ejercicio 71

Enunciado: Crea una clase `VehiculoConVelocidad` que herede de `Vehiculo` y agregue un método `acelerar` que aumente la velocidad actual.
Explicación: Introduce métodos que modifican atributos de la subclase.

```
class Vehiculo:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

class VehiculoConVelocidad(Vehiculo):
    def __init__(self, marca, modelo, velocidad=0):
        super().__init__(marca, modelo)
        self.velocidad = velocidad

    def acelerar(self, incremento):
        self.velocidad += incremento

auto = VehiculoConVelocidad("Toyota", "Corolla")
auto.acelerar(30)
print(auto.marca, auto.modelo, "Velocidad:", auto.velocidad)
```

Ejercicio 72

Enunciado: Crea una clase `PerroConEdadYColor` que herede de `Perro` y agregue métodos para cambiar el color y la edad.
Explicación: Introduce métodos que actualizan atributos de la subclase.

```
class Perro:
    def __init__(self, nombre):
        self.nombre = nombre

class PerroConEdadYColor(Perro):
    def __init__(self, nombre, edad, color):
        super().__init__(nombre)
        self.edad = edad
        self.color = color

    def cambiar_edad(self, nueva_edad):
        self.edad = nueva_edad

    def cambiar_color(self, nuevo_color):
        self.color = nuevo_color

perro = PerroConEdadYColor("Max", 3, "Negro")
perro.cambiar_edad(4)
perro.cambiar_color("Marrón")
print(perro.nombre, perro.edad, perro.color)
```

Ejercicio 73

Enunciado: Crea una clase `CuentaConRetiroSeguro` que herede de `CuentaBancaria` y permita retiros solo si el saldo no baja de 100.
Explicación: Introduce métodos con condiciones de control sobre atributos.

```
class CuentaBancaria:
    def __init__(self, titular, saldo):
        self.titular = titular
        self.saldo = saldo

class CuentaConRetiroSeguro(CuentaBancaria):
    def retirar(self, cantidad):
        if self.saldo - cantidad >= 100:
            self.saldo -= cantidad
        else:
            print("Retiro no permitido, saldo mínimo 100")

cuenta = CuentaConRetiroSeguro("Ana", 500)
cuenta.retirar(450)
print(cuenta.saldo)
cuenta.retirar(300)
```

Ejercicio 74

Enunciado: Crea una clase `LibroConResenas` que herede de `Libro` y agregue un atributo lista `resenas` y un método para añadir reseñas.
Explicación: Introduce métodos que gestionan listas de atributos.

```
class Libro:
    def __init__(self, titulo, autor):
        self.titulo = titulo
        self.autor = autor

class LibroConResenas(Libro):
    def __init__(self, titulo, autor):
        super().__init__(titulo, autor)
        self.resenas = []

    def agregar_resena(self, resena):
        self.resenas.append(resena)

libro = LibroConResenas("1984", "George Orwell")
libro.agregar_resena("Excelente libro")
libro.agregar_resena("Muy recomendable")
print(libro.titulo, libro.autor, libro.resenas)
```

Ejercicio 75

Enunciado: Crea una clase `EmpleadoConProyecto` que herede de `Empleado` y agregue un atributo lista `proyectos` y un método para añadir proyectos.
Explicación: Introduce atributos tipo lista y métodos para gestionarlos en subclases.

```
class Empleado:
    def __init__(self, nombre, sueldo):
        self.nombre = nombre
        self.sueldo = sueldo

class EmpleadoConProyecto(Empleado):
    def __init__(self, nombre, sueldo):
        super().__init__(nombre, sueldo)
        self.proyectos = []

    def agregar_proyecto(self, proyecto):
        self.proyectos.append(proyecto)

empleado = EmpleadoConProyecto("Luis", 2000)
empleado.agregar_proyecto("Proyecto A")
empleado.agregar_proyecto("Proyecto B")
print(empleado.nombre, empleado.proyectos)
```

Ejercicio 76

Enunciado: Crea una clase `EstudianteConPromedioCondicional` que herede de `Estudiante` y agregue un método que indique si aprueba (promedio ≥6).
Explicación: Introduce métodos que evalúan condiciones sobre atributos.

```
class Estudiante:
    def __init__(self, nombre, notas):
        self.nombre = nombre
        self.notas = notas

class EstudianteConPromedioCondicional(Estudiante):
    def aprobar(self):
        return sum(self.notas)/len(self.notas) >= 6

estudiante = EstudianteConPromedioCondicional("Sara", [7,5,8])
print(estudiante.nombre, "Aprueba:", estudiante.aprobar())
```

Ejercicio 77

Enunciado: Crea una clase `SmartphoneConAlmacenamientoApps` que herede de `Smartphone` y permita instalar apps solo si hay espacio disponible.
Explicación: Introduce métodos con condiciones y gestión de atributos tipo lista y numéricos.

```
class Smartphone:
    def __init__(self, marca, modelo, almacenamiento=100):
        self.marca = marca
        self.modelo = modelo
        self.almacenamiento = almacenamiento
        self.apps = []

class SmartphoneConAlmacenamientoApps(Smartphone):
    def instalar_app(self, app, tamaño):
        if self.almacenamiento - tamaño >= 0:
            self.apps.append(app)
            self.almacenamiento -= tamaño
        else:
            print("No hay espacio suficiente")

telefono = SmartphoneConAlmacenamientoApps("Samsung", "S21")
telefono.instalar_app("WhatsApp", 5)
telefono.instalar_app("Spotify", 10)
print(telefono.apps, "Espacio disponible:",
telefono.almacenamiento)
```

Ejercicio 78

Enunciado: Crea una clase `BicicletaConVelocidadCambio` que herede de `Bicicleta` y permita cambiar la marcha y mostrar la velocidad actual.
Explicación: Introduce métodos que modifican y muestran atributos de la subclase.

```
class Bicicleta:
    def __init__(self, marca, tipo):
        self.marca = marca
        self.tipo = tipo

class BicicletaConVelocidadCambio(Bicicleta):
    def __init__(self, marca, tipo, numero_marchas, velocidad=0):
        super().__init__(marca, tipo)
        self.numero_marchas = numero_marchas
        self.velocidad = velocidad

    def cambiar_marcha(self, nueva_marcha):
        self.numero_marchas = nueva_marcha

    def mostrar_velocidad(self):
        print(f"Velocidad: {self.velocidad} km/h")

bici = BicicletaConVelocidadCambio("Trek", "Montaña", 21)
bici.cambiar_marcha(18)
bici.velocidad = 25
bici.mostrar_velocidad()
```

Ejercicio 79

Enunciado: Crea una clase `AlumnoConNotasHistorial` que herede de `Alumno` y permita registrar notas con fecha en un diccionario.
Explicación: Introduce atributos tipo diccionario y métodos que los gestionan.

```
class Alumno:
    def __init__(self, nombre):
        self.nombre = nombre

class AlumnoConNotasHistorial(Alumno):
    def __init__(self, nombre):
        super().__init__(nombre)
        self.historial = {}

    def agregar_nota(self, fecha, nota):
        self.historial[fecha] = nota

alumno = AlumnoConNotasHistorial("Lucía")
alumno.agregar_nota("2025-10-26", 9)
alumno.agregar_nota("2025-10-27", 8)
print(alumno.nombre, alumno.historial)
```

Ejercicio 80

Enunciado: Crea una clase `ProfesorConClases` que herede de `Profesor` y permita agregar nuevas clases a su lista de asignaturas.
Explicación: Introduce métodos que gestionan listas de atributos en subclases.

```
class Profesor:
    def __init__(self, nombre):
        self.nombre = nombre

class ProfesorConClases(Profesor):
    def __init__(self, nombre):
        super().__init__(nombre)
        self.clases = []

    def agregar_clase(self, clase):
        self.clases.append(clase)

profesor = ProfesorConClases("Carlos")
profesor.agregar_clase("Matemáticas")
profesor.agregar_clase("Física")
print(profesor.nombre, profesor.clases)
```

Ejercicio 81

Enunciado: Crea una clase `VehiculoConColor` que herede de `Vehiculo` y agregue un atributo `color` y un método para cambiarlo.
Explicación: Introduce herencia y métodos que actualizan atributos de la subclase.

```
class Vehiculo:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

class VehiculoConColor(Vehiculo):
    def __init__(self, marca, modelo, color):
        super().__init__(marca, modelo)
        self.color = color

    def cambiar_color(self, nuevo_color):
        self.color = nuevo_color

auto = VehiculoConColor("Honda", "Civic", "Rojo")
auto.cambiar_color("Azul")
print(auto.marca, auto.modelo, auto.color)
```

Ejercicio 82

Enunciado: Crea una clase `PerroConPeso` que herede de `Perro` y agregue un atributo `peso` y un método para modificarlo.
Explicación: Practica herencia y actualización de atributos numéricos de la subclase.

```
class Perro:
    def __init__(self, nombre):
        self.nombre = nombre

class PerroConPeso(Perro):
    def __init__(self, nombre, peso):
        super().__init__(nombre)
        self.peso = peso

    def cambiar_peso(self, nuevo_peso):
        self.peso = nuevo_peso

perro = PerroConPeso("Max", 10)
perro.cambiar_peso(12)
print(perro.nombre, perro.peso)
```

Ejercicio 83

Enunciado: Crea una clase `CuentaConBloqueo` que herede de `CuentaBancaria` y agregue un método para bloquear la cuenta.
Explicación: Introduce métodos que cambian el estado booleano de atributos de la subclase.

```
class CuentaBancaria:
    def __init__(self, titular, saldo):
        self.titular = titular
        self.saldo = saldo

class CuentaConBloqueo(CuentaBancaria):
    def __init__(self, titular, saldo):
        super().__init__(titular, saldo)
        self.bloqueada = False

    def bloquear(self):
        self.bloqueada = True

cuenta = CuentaConBloqueo("Ana", 1000)
cuenta.bloquear()
print(cuenta.titular, cuenta.saldo, "Bloqueada:",
cuenta.bloqueada)
```

Ejercicio 84

Enunciado: Crea una clase `LibroConDisponibilidad` que herede de `Libro` y agregue un método que indique si está disponible para préstamo.
Explicación: Introduce métodos que verifican atributos booleanos y devuelven información.

```
class Libro:
    def __init__(self, titulo, autor):
        self.titulo = titulo
        self.autor = autor
        self.prestado = False

class LibroConDisponibilidad(Libro):
    def disponible(self):
        return not self.prestado

libro = LibroConDisponibilidad("1984", "George Orwell")
print(libro.titulo, libro.disponible())
libro.prestado = True
print(libro.titulo, libro.disponible())
```

Ejercicio 85

Enunciado: Crea una clase `EmpleadoConHorasExtras` que herede de `Empleado` y agregue un método que calcule el sueldo total sumando horas extras.
Explicación: Introduce métodos que realizan cálculos usando atributos heredados y parámetros adicionales.

```
class Empleado:
    def __init__(self, nombre, sueldo):
        self.nombre = nombre
        self.sueldo = sueldo

class EmpleadoConHorasExtras(Empleado):
    def calcular_sueldo_total(self, horas_extras, valor_por_hora):
        return self.sueldo + horas_extras * valor_por_hora

empleado = EmpleadoConHorasExtras("Luis", 2000)
print(empleado.nombre, "Sueldo total:",
empleado.calcular_sueldo_total(10, 15))
```

Ejercicio 86

Enunciado: Crea una clase `EstudianteConMaterias` que herede de `Estudiante` y agregue un método para registrar materias aprobadas.
Explicación: Introduce métodos que gestionan listas de atributos de la subclase.

```
class Estudiante:
    def __init__(self, nombre):
        self.nombre = nombre

class EstudianteConMaterias(Estudiante):
    def __init__(self, nombre):
        super().__init__(nombre)
        self.materias_aprobadas = []

    def agregar_materia(self, materia):
        self.materias_aprobadas.append(materia)

estudiante = EstudianteConMaterias("Sara")
estudiante.agregar_materia("Matemáticas")
estudiante.agregar_materia("Física")
print(estudiante.nombre, estudiante.materias_aprobadas)
```

Ejercicio 87

Enunciado: Crea una clase `SmartphoneConBateriaYApps` que herede de `Smartphone` y permita instalar apps solo si la batería es suficiente.
Explicación: Introduce métodos que evalúan condiciones y modifican listas y atributos numéricos.

```
class Smartphone:
    def __init__(self, marca, modelo, bateria=100):
        self.marca = marca
        self.modelo = modelo
        self.bateria = bateria
        self.apps = []

class SmartphoneConBateriaYApps(Smartphone):
    def instalar_app(self, app, consumo_bateria):
        if self.bateria - consumo_bateria >= 0:
            self.apps.append(app)
            self.bateria -= consumo_bateria
        else:
            print("Batería insuficiente")

telefono = SmartphoneConBateriaYApps("Apple", "iPhone 14")
telefono.instalar_app("WhatsApp", 5)
telefono.instalar_app("Spotify", 10)
print(telefono.apps, "Batería restante:", telefono.bateria)
```

Ejercicio 88

Enunciado: Crea una clase `BicicletaConKilometraje` que herede de `Bicicleta` y agregue un método para sumar kilómetros recorridos.
Explicación: Introduce métodos que incrementan atributos numéricos de la subclase.

```
class Bicicleta:
    def __init__(self, marca, tipo):
        self.marca = marca
        self.tipo = tipo

class BicicletaConKilometraje(Bicicleta):
    def __init__(self, marca, tipo):
        super().__init__(marca, tipo)
        self.kilometros = 0

    def recorrer(self, km):
        self.kilometros += km

bici = BicicletaConKilometraje("Trek", "Montaña")
bici.recorrer(15)
bici.recorrer(10)
print(bici.marca, bici.tipo, "Kilómetros:", bici.kilometros)
```

Ejercicio 89

Enunciado: Crea una clase `AlumnoConNotasExtra` que herede de `Alumno` y agregue un método para calcular promedio considerando un bonus adicional.
Explicación: Introduce métodos que realizan cálculos con atributos heredados y parámetros adicionales.

```
class Alumno:
    def __init__(self, nombre, notas):
        self.nombre = nombre
        self.notas = notas

class AlumnoConNotasExtra(Alumno):
    def promedio_con_bonus(self, bonus):
        return sum(self.notas)/len(self.notas) + bonus

alumno = AlumnoConNotasExtra("Lucía", [7,8,9])
print(alumno.nombre, "Promedio con bonus:",
alumno.promedio_con_bonus(1))
```

Ejercicio 90

Enunciado: Crea una clase `ProfesorConClasesYEdad` que herede de `Profesor` y permita agregar clases y mostrar la edad.
Explicación: Combina métodos para listas y atributos numéricos en subclases.

```
class Profesor:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

class ProfesorConClasesYEdad(Profesor):
    def __init__(self, nombre, edad):
        super().__init__(nombre, edad)
        self.clases = []

    def agregar_clase(self, clase):
        self.clases.append(clase)

profesor = ProfesorConClasesYEdad("Carlos", 45)
profesor.agregar_clase("Matemáticas")
profesor.agregar_clase("Física")
print(profesor.nombre, profesor.edad, profesor.clases)
```

Ejercicio 91

Enunciado: Crea una clase `VehiculoConLuces` que herede de `Vehiculo` y agregue métodos para encender y apagar las luces.
Explicación: Introduce métodos que cambian el estado booleano de atributos de la subclase.

```
class Vehiculo:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

class VehiculoConLuces(Vehiculo):
    def __init__(self, marca, modelo):
        super().__init__(marca, modelo)
        self.luces_encendidas = False

    def encender_luces(self):
        self.luces_encendidas = True

    def apagar_luces(self):
        self.luces_encendidas = False

auto = VehiculoConLuces("Ford", "Focus")
auto.encender_luces()
print(auto.marca, auto.modelo, "Luces encendidas:", auto.luces_encendidas)
auto.apagar_luces()
print("Luces encendidas:", auto.luces_encendidas)
```

Ejercicio 92

Enunciado: Crea una clase `PerroConEnergia` que herede de `Perro` y agregue un método `jugar` que disminuya su energía.
Explicación: Introduce métodos que modifican atributos numéricos de la subclase.

```
class Perro:
    def __init__(self, nombre):
        self.nombre = nombre


class PerroConEnergia(Perro):
    def __init__(self, nombre, energia):
        super().__init__(nombre)
        self.energia = energia

    def jugar(self, tiempo):
        self.energia -= tiempo

perro = PerroConEnergia("Max", 10)
perro.jugar(3)
print(perro.nombre, "Energía restante:", perro.energia)
```

Ejercicio 93

Enunciado: Crea una clase `CuentaConDepositoCondicional` que herede de `CuentaBancaria` y solo permita depósitos si la cantidad es positiva.
Explicación: Introduce métodos con condiciones que verifican valores antes de modificar atributos.

```
class CuentaBancaria:
    def __init__(self, titular, saldo):
        self.titular = titular
        self.saldo = saldo

class CuentaConDepositoCondicional(CuentaBancaria):
    def depositar(self, cantidad):
        if cantidad > 0:
            self.saldo += cantidad
        else:
            print("Cantidad inválida")

cuenta = CuentaConDepositoCondicional("Ana", 1000)
cuenta.depositar(200)
cuenta.depositar(-50)
print(cuenta.saldo)
```

Ejercicio 94

Enunciado: Crea una clase `LibroConPaginasLeidas` que herede de `Libro` y agregue un método para leer páginas y actualizar un atributo de seguimiento.
Explicación: Introduce métodos que incrementan atributos de control.

```
class Libro:
    def __init__(self, titulo, autor, paginas):
        self.titulo = titulo
        self.autor = autor
        self.paginas = paginas
        self.paginas_leidas = 0

class LibroConPaginasLeidas(Libro):
    def leer(self, paginas):
        self.paginas_leidas += paginas
        if self.paginas_leidas > self.paginas:
            self.paginas_leidas = self.paginas

libro = LibroConPaginasLeidas("1984", "George Orwell", 300)
libro.leer(50)
libro.leer(100)
print(libro.titulo, "Páginas leídas:", libro.paginas_leidas)
```

Ejercicio 95

Enunciado: Crea una clase `EmpleadoConBonificacion` que herede de `Empleado` y agregue un método que aumente el sueldo según un porcentaje de bonificación.
Explicación: Introduce métodos que calculan valores a partir de atributos heredados y parámetros.

```
class Empleado:
    def __init__(self, nombre, sueldo):
        self.nombre = nombre
        self.sueldo = sueldo

class EmpleadoConBonificacion(Empleado):
    def aplicar_bonificacion(self, porcentaje):
        self.sueldo += self.sueldo * porcentaje / 100

empleado = EmpleadoConBonificacion("Luis", 2000)
empleado.aplicar_bonificacion(10)
print(empleado.nombre, "Sueldo con bonificación:", empleado.sueldo)
```

Ejercicio 96

Enunciado: Crea una clase `EstudianteConRegistroNotas` que herede de `Estudiante` y permita agregar y mostrar todas las notas.
Explicación: Introduce métodos que gestionan listas de atributos.

```
class Estudiante:
    def __init__(self, nombre):
        self.nombre = nombre


class EstudianteConRegistroNotas(Estudiante):
    def __init__(self, nombre):
        super().__init__(nombre)
        self.notas = []

    def agregar_nota(self, nota):
        self.notas.append(nota)

    def mostrar_notas(self):
        print(self.notas)

estudiante = EstudianteConRegistroNotas("Sara")
estudiante.agregar_nota(7)
estudiante.agregar_nota(8)
print(estudiante.nombre, end=" Notas: ")
estudiante.mostrar_notas()
```

Ejercicio 97

Enunciado: Crea una clase `SmartphoneConCarga` que herede de `Smartphone` y agregue un método para cargar la batería.
Explicación: Introduce métodos que modifican atributos numéricos de la subclase.

```
class Smartphone:
    def __init__(self, marca, modelo, bateria=50):
        self.marca = marca
        self.modelo = modelo
        self.bateria = bateria

class SmartphoneConCarga(Smartphone):
    def cargar(self, cantidad):
        self.bateria += cantidad
        if self.bateria > 100:
            self.bateria = 100

telefono = SmartphoneConCarga("Apple", "iPhone 14")
telefono.cargar(30)
print(telefono.marca, telefono.modelo, "Batería:", telefono.bateria)
```

Ejercicio 98

Enunciado: Crea una clase `BicicletaConRuta` que herede de `Bicicleta` y agregue un método para registrar rutas recorridas.
Explicación: Introduce métodos que gestionan listas de atributos de la subclase.

```
class Bicicleta:
    def __init__(self, marca, tipo):
        self.marca = marca
        self.tipo = tipo

class BicicletaConRuta(Bicicleta):
    def __init__(self, marca, tipo):
        super().__init__(marca, tipo)
        self.rutas = []

    def agregar_ruta(self, ruta):
        self.rutas.append(ruta)

bici = BicicletaConRuta("Trek", "Montaña")
bici.agregar_ruta("Parque Central")
bici.agregar_ruta("Ciudad")
print(bici.marca, bici.tipo, "Rutas:", bici.rutas)
```

Ejercicio 99

Enunciado: Crea una clase `AlumnoConPromocion` que herede de `Alumno` y agregue un método que indique si el alumno puede ser promovido según promedio ≥6.
Explicación: Introduce métodos que evalúan condiciones sobre atributos de la subclase.

```
class Alumno:
    def __init__(self, nombre, notas):
        self.nombre = nombre
        self.notas = notas

class AlumnoConPromocion(Alumno):
    def promocion(self):
        return sum(self.notas)/len(self.notas) >= 6

alumno = AlumnoConPromocion("Lucía", [7,8,6])
print(alumno.nombre, "Promoción:", alumno.promocion())
```

Ejercicio 100

Enunciado: Crea una clase `ProfesorConSalarioYAsignaturas` que herede de `Profesor` y agregue métodos para aumentar el salario y agregar asignaturas.
Explicación: Combina métodos que modifican atributos numéricos y listas en subclases.

```
class Profesor:
    def __init__(self, nombre, salario):
        self.nombre = nombre
        self.salario = salario
        self.asignaturas = []

class ProfesorConSalarioYAsignaturas(Profesor):
    def aumentar_salario(self, porcentaje):
        self.salario += self.salario * porcentaje / 100

    def agregar_asignatura(self, asignatura):
        self.asignaturas.append(asignatura)

profesor = ProfesorConSalarioYAsignaturas("Carlos", 3000)
profesor.aumentar_salario(10)
profesor.agregar_asignatura("Matemáticas")
profesor.agregar_asignatura("Física")
print(profesor.nombre, profesor.salario, profesor.asignaturas)
```

Ejercicios de python fechas y hora

Ejercicio 1

Enunciado: Muestra la fecha y hora actual.
Explicación: Usamos el módulo `datetime` para obtener la fecha y hora actuales.

```
from datetime import datetime

ahora = datetime.now()
print("Fecha y hora actual:", ahora)
```

Ejercicio 2

Enunciado: Muestra solo la fecha actual.
Explicación: `datetime.date()` extrae solo la parte de la fecha.

```
from datetime import datetime

hoy = datetime.now().date()
print("Fecha actual:", hoy)
```

Ejercicio 3

Enunciado: Muestra solo la hora actual.
Explicación: `datetime.time()` extrae solo la parte de la hora.

```
from datetime import datetime

hora_actual = datetime.now().time()
print("Hora actual:", hora_actual)
```

Ejercicio 4

Enunciado: Muestra el año, mes y día por separado.
Explicación: `datetime` permite acceder a cada componente de la fecha.

```
from datetime import datetime

ahora = datetime.now()
print("Año:", ahora.year)
print("Mes:", ahora.month)
print("Día:", ahora.day)
```

Ejercicio 5

Enunciado: Muestra la hora, minuto y segundo por separado.
Explicación: `datetime` permite acceder a cada componente de la hora.

```
from datetime import datetime

ahora = datetime.now()
print("Hora:", ahora.hour)
print("Minuto:", ahora.minute)
print("Segundo:", ahora.second)
```

Ejercicio 6

Enunciado: Formatea la fecha en formato DD/MM/AAAA.
Explicación: `strftime` permite convertir fechas a cadenas con formato personalizado.

```
from datetime import datetime

ahora = datetime.now()
fecha_formateada = ahora.strftime("%d/%m/%Y")
print("Fecha formateada:", fecha_formateada)
```

Ejercicio 7

Enunciado: Formatea la hora en formato HH:MM:SS.
Explicación: `strftime` permite convertir la hora en cadena legible.

```
from datetime import datetime

ahora = datetime.now()
hora_formateada = ahora.strftime("%H:%M:%S")
print("Hora formateada:", hora_formateada)
```

Ejercicio 8

Enunciado: Obtén el día de la semana actual.
Explicación: `strftime("%A")` devuelve el nombre del día de la semana.

```
from datetime import datetime

ahora = datetime.now()
dia_semana = ahora.strftime("%A")
print("Día de la semana:", dia_semana)
```

Ejercicio 9

Enunciado: Suma 5 días a la fecha actual.
Explicación: `timedelta` permite realizar operaciones con fechas.

```
from datetime import datetime, timedelta

hoy = datetime.now()
futuro = hoy + timedelta(days=5)
print("Fecha actual:", hoy.date())
print("Fecha dentro de 5 días:", futuro.date())
```

Ejercicio 10

Enunciado: Resta 10 días a la fecha actual.
Explicación: `timedelta` permite restar días a una fecha.

```
from datetime import datetime, timedelta

hoy = datetime.now()
pasado = hoy - timedelta(days=10)
print("Fecha actual:", hoy.date())
print("Fecha hace 10 días:", pasado.date())
```

Ejercicio 11

Enunciado: Calcula la diferencia en días entre dos fechas.
Explicación: Usamos `timedelta` para calcular la diferencia entre fechas.

```
from datetime import datetime


fecha1 = datetime(2025, 10, 26)

fecha2 = datetime(2025, 11, 5)

diferencia = fecha2 - fecha1

print("Diferencia en días:", diferencia.days)
```

Ejercicio 12

Enunciado: Calcula cuántos segundos han pasado desde medianoche.
Explicación: Restamos la hora actual con la medianoche y usamos `total_seconds()`.

```
from datetime import datetime, time

ahora = datetime.now()
medianoche = datetime.combine(ahora.date(), time())
segundos = (ahora - medianoche).total_seconds()
print("Segundos desde medianoche:", segundos)
```

Ejercicio 13

Enunciado: Convierte una cadena con fecha en objeto datetime.
Explicación: `strptime` permite convertir cadenas en fechas con formato definido.

```
from datetime import datetime

fecha_str = "26/10/2025"
fecha_obj = datetime.strptime(fecha_str, "%d/%m/%Y")
print("Fecha como objeto datetime:", fecha_obj)
```

Ejercicio 14

Enunciado: Convierte un objeto datetime a cadena en formato "AAAA-MM-DD".
Explicación: `strftime` permite formatear fechas en cadenas específicas.

```
from datetime import datetime

ahora = datetime.now()
fecha_formateada = ahora.strftime("%Y-%m-%d")
print("Fecha formateada:", fecha_formateada)
```

Ejercicio 15

Enunciado: Muestra la fecha y hora actuales en formato completo legible.
Explicación: `strftime` con formato largo muestra día de la semana, fecha y hora.

```
from datetime import datetime


ahora = datetime.now()
fecha_hora_completa = ahora.strftime("%A, %d %B %Y %H:%M:%S")
print("Fecha y hora completa:", fecha_hora_completa)
```

Ejercicio 16

Enunciado: Obtén el número de semana del año actual.
Explicación: `%U` en `strftime` devuelve la semana del año (domingo como primer día).

```
from datetime import datetime


ahora = datetime.now()
semana_año = ahora.strftime("%U")
print("Número de semana del año:", semana_año)
```

Ejercicio 17

Enunciado: Muestra el nombre abreviado del mes actual.
Explicación: `%b` en `strftime` devuelve el mes abreviado.

```
from datetime import datetime


ahora = datetime.now()
mes_abreviado = ahora.strftime("%b")
print("Mes abreviado:", mes_abreviado)
```

Ejercicio 18

Enunciado: Muestra el año y la semana ISO del año actual.
Explicación: `isocalendar()` devuelve una tupla (año, semana, día de la semana).

```
from datetime import datetime


ahora = datetime.now()

iso = ahora.isocalendar()

print("Año ISO:", iso[0], "Semana ISO:", iso[1], "Día ISO:",
iso[2])
```

Ejercicio 19

Enunciado: Calcula la fecha dentro de 2 semanas a partir de hoy.
Explicación: Usamos `timedelta` para sumar días a la fecha actual.

```
from datetime import datetime, timedelta


hoy = datetime.now()

dentro_2_semanas = hoy + timedelta(weeks=2)

print("Fecha dentro de 2 semanas:", dentro_2_semanas.date())
```

Ejercicio 20

Enunciado: Calcula cuántos días faltan para fin de año.
Explicación: Restamos la fecha actual con el 31 de diciembre del mismo año.

```
from datetime import datetime


hoy = datetime.now()

fin_de_año = datetime(hoy.year, 12, 31)

faltan_dias = (fin_de_año - hoy).days

print("Días para fin de año:", faltan_dias)
```

Ejercicio 21

Enunciado: Muestra si el año actual es bisiesto o no.
Explicación: Usamos el módulo `calendar` para verificar años bisiestos.

```
import calendar
from datetime import datetime

año_actual = datetime.now().year
es_bisiesto = calendar.isleap(año_actual)
print("Año actual:", año_actual, "Bisiesto:", es_bisiesto)
```

Ejercicio 22

Enunciado: Calcula la fecha de nacimiento a partir de la edad.
Explicación: Restamos años usando `timedelta` aproximado (365 días por año).

```
from datetime import datetime, timedelta

edad = 25
hoy = datetime.now()
fecha_nacimiento = hoy - timedelta(days=edad*365)
print("Fecha aproximada de nacimiento:", fecha_nacimiento.date())
```

Ejercicio 23

Enunciado: Obtén la hora en formato 12 horas con AM/PM.
Explicación: `strftime("%I:%M %p")` devuelve la hora en formato 12 horas.

```
from datetime import datetime

ahora = datetime.now()
hora_12 = ahora.strftime("%I:%M %p")
print("Hora en formato 12 horas:", hora_12)
```

Ejercicio 24

Enunciado: Calcula cuántos días tiene el mes actual.
Explicación: Usamos `calendar.monthrange` para obtener el número de días del mes.

```
import calendar
from datetime import datetime

ahora = datetime.now()
dias_mes = calendar.monthrange(ahora.year, ahora.month)[1]
print("Días del mes actual:", dias_mes)
```

Ejercicio 25

Enunciado: Muestra el primer día de la semana del mes actual.
Explicación: `calendar.monthrange` devuelve el primer día de la semana del mes.

```
import calendar
from datetime import datetime

ahora = datetime.now()
primer_dia_semana = calendar.monthrange(ahora.year, ahora.month)
[0]
print("Primer día de la semana del mes actual (0=lunes):",
primer_dia_semana)
```

Ejercicio 26

Enunciado: Muestra cuántos días faltan para tu próximo cumpleaños.
Explicación: Creamos una fecha con el año actual y restamos la fecha actual.

```
from datetime import datetime

hoy = datetime.now()
cumpleaños = datetime(hoy.year, 12, 15)  # ejemplo 15 de diciembre
if cumpleaños < hoy:
    cumpleaños = datetime(hoy.year + 1, 12, 15)
faltan_dias = (cumpleaños - hoy).days
print("Días para el próximo cumpleaños:", faltan_dias)
```

Ejercicio 27

Enunciado: Muestra la fecha de ayer.
Explicación: Usamos `timedelta(days=1)` para restar un día a la fecha actual.

```
from datetime import datetime, timedelta


hoy = datetime.now()
ayer = hoy - timedelta(days=1)
print("Fecha de ayer:", ayer.date())
```

Ejercicio 28

Enunciado: Muestra la fecha de mañana.
Explicación: Usamos `timedelta(days=1)` para sumar un día a la fecha actual.

```
from datetime import datetime, timedelta


hoy = datetime.now()
mañana = hoy + timedelta(days=1)
print("Fecha de mañana:", mañana.date())
```

Ejercicio 29

Enunciado: Calcula cuántos segundos faltan para fin de día.
Explicación: Restamos la hora actual a la medianoche del día siguiente.

```
from datetime import datetime, timedelta, time


ahora = datetime.now()
medianoche_siguiente = datetime.combine(ahora.date() +
timedelta(days=1), time())
segundos_restantes = (medianoche_siguiente -
ahora).total_seconds()
print("Segundos para fin de día:", segundos_restantes)
```

Ejercicio 30

Enunciado: Muestra la fecha y hora en formato ISO.
Explicación: `isoformat()` devuelve la fecha y hora en formato estándar ISO 8601.

```
from datetime import datetime


ahora = datetime.now()
iso = ahora.isoformat()
print("Fecha y hora en formato ISO:", iso)
```

Ejercicio 31

Enunciado: Muestra el número del día del año actual.
Explicación: `timetuple().tm_yday` devuelve el número de día dentro del año.

```
from datetime import datetime


hoy = datetime.now()
dia_del_año = hoy.timetuple().tm_yday
print("Día del año:", dia_del_año)
```

Ejercicio 32

Enunciado: Muestra el número de la semana del año (lunes como primer día).
Explicación: `isocalendar()[1]` devuelve la semana ISO del año.

```
from datetime import datetime


hoy = datetime.now()
semana_iso = hoy.isocalendar()[1]
print("Semana del año (ISO):", semana_iso)
```

Ejercicio 33

Enunciado: Muestra cuántos segundos han pasado desde el inicio del año.
Explicación: Restamos la fecha actual con el 1 de enero y usamos `total_seconds()`.

```
from datetime import datetime

hoy = datetime.now()
inicio_año = datetime(hoy.year, 1, 1)
segundos = (hoy - inicio_año).total_seconds()
print("Segundos desde inicio del año:", int(segundos))
```

Ejercicio 34

Enunciado: Suma 3 horas a la hora actual.
Explicación: `timedelta(hours=3)` permite agregar horas a la fecha y hora actual.

```
from datetime import datetime, timedelta

ahora = datetime.now()
nueva_hora = ahora + timedelta(hours=3)
print("Hora actual más 3 horas:", nueva_hora)
```

Ejercicio 35

Enunciado: Resta 45 minutos a la hora actual.
Explicación: `timedelta(minutes=45)` permite restar minutos a la fecha y hora actual.

```
from datetime import datetime, timedelta

ahora = datetime.now()
menos_45_min = ahora - timedelta(minutes=45)
print("Hora actual menos 45 minutos:", menos_45_min)
```

Ejercicio 36

Enunciado: Convierte segundos a horas, minutos y segundos.
Explicación: Calculamos horas, minutos y segundos a partir de un total de segundos.

```
segundos_totales = 3675  # ejemplo

horas = segundos_totales // 3600

minutos = (segundos_totales % 3600) // 60

segundos = segundos_totales % 60

print(f"{segundos_totales} segundos son {horas}h {minutos}m {segundos}s")
```

Ejercicio 37

Enunciado: Muestra el mes actual en número y nombre.
Explicación: `strftime("%B")` devuelve el nombre completo del mes.

```
from datetime import datetime


hoy = datetime.now()

print("Mes en número:", hoy.month)

print("Mes en nombre:", hoy.strftime("%B"))
```

Ejercicio 38

Enunciado: Calcula cuántos días tiene febrero de un año dado.
Explicación: `calendar.monthrange` devuelve la cantidad de días en un mes específico.

```
import calendar


año = 2024

dias_febrero = calendar.monthrange(año, 2)[1]

print(f"Febrero de {año} tiene {dias_febrero} días")
```

Ejercicio 39

Enunciado: Calcula cuántos días faltan para el próximo lunes.
Explicación: Restamos el día actual al lunes siguiente usando `timedelta`.

```
from datetime import datetime, timedelta


hoy = datetime.now()
dias_para_lunes = (7 - hoy.weekday()) % 7
if dias_para_lunes == 0:
    dias_para_lunes = 7
proximo_lunes = hoy + timedelta(days=dias_para_lunes)
print("Días para el próximo lunes:", dias_para_lunes)
print("Fecha del próximo lunes:", proximo_lunes.date())
```

Ejercicio 40

Enunciado: Muestra la fecha y hora actual con milisegundos.
Explicación: `strftime("%f")` devuelve los microsegundos; podemos mostrar milisegundos dividiendo entre 1000.

```
from datetime import datetime


ahora = datetime.now()
milisegundos = ahora.microsecond // 1000
print("Fecha y hora con milisegundos:", ahora.strftime("%Y-%m-%d %H:%M:%S"), milisegundos, "ms")
```

Ejercicio 41

Enunciado: Muestra el nombre del día anterior.
Explicación: Restamos un día a la fecha actual y usamos `strftime("%A")` para obtener el nombre del día.

```
from datetime import datetime, timedelta


hoy = datetime.now()
ayer = hoy - timedelta(days=1)
print("Nombre del día anterior:", ayer.strftime("%A"))
```

Ejercicio 42

Enunciado: Muestra el nombre del día siguiente.
Explicación: Sumamos un día a la fecha actual y usamos `strftime("%A")`.

```
from datetime import datetime, timedelta


hoy = datetime.now()
mañana = hoy + timedelta(days=1)
print("Nombre del día siguiente:", mañana.strftime("%A"))
```

Ejercicio 43

Enunciado: Convierte un objeto `date` en cadena con formato "DD-MM-AAAA".
Explicación: `strftime` permite formatear la fecha a cadena.

```
from datetime import date


hoy = date.today()
fecha_formateada = hoy.strftime("%d-%m-%Y")
print("Fecha formateada:", fecha_formateada)
```

Ejercicio 44

Enunciado: Convierte una cadena "AAAA/MM/DD" en objeto `date`.
Explicación: `strptime` permite crear un objeto de fecha a partir de una cadena.

```
from datetime import datetime


fecha_str = "2025/10/26"
fecha_obj = datetime.strptime(fecha_str, "%Y/%m/%d").date()
print("Fecha como objeto date:", fecha_obj)
```

Ejercicio 45

Enunciado: Calcula cuántos días han pasado desde tu fecha de nacimiento.
Explicación: Restamos la fecha actual con la fecha de nacimiento usando `timedelta`.

```
from datetime import datetime


fecha_nacimiento = datetime(2000, 5, 15)
hoy = datetime.now()
dias_transcurridos = (hoy - fecha_nacimiento).days
print("Días desde nacimiento:", dias_transcurridos)
```

Ejercicio 46

Enunciado: Suma 90 minutos a la hora actual.
Explicación: `timedelta(minutes=90)` permite sumar minutos a la fecha y hora actual.

```
from datetime import datetime, timedelta


ahora = datetime.now()
nueva_hora = ahora + timedelta(minutes=90)
print("Hora actual más 90 minutos:", nueva_hora)
```

Ejercicio 47

Enunciado: Resta 2 horas y 30 minutos a la hora actual.
Explicación: `timedelta(hours=2, minutes=30)` permite restar horas y minutos a la fecha y hora actual.

```
from datetime import datetime, timedelta


ahora = datetime.now()
hora_modificada = ahora - timedelta(hours=2, minutes=30)
print("Hora actual menos 2h30m:", hora_modificada)
```

Ejercicio 48

Enunciado: Muestra la fecha y hora actuales en formato personalizado "DD/MM/AAAA HH:MM".
Explicación: `strftime` permite personalizar el formato de fecha y hora.

```
from datetime import datetime


ahora = datetime.now()
fecha_hora = ahora.strftime("%d/%m/%Y %H:%M")
print("Fecha y hora personalizada:", fecha_hora)
```

Ejercicio 49

Enunciado: Calcula el tiempo transcurrido en segundos entre dos fechas.
Explicación: Restamos dos objetos `datetime` y usamos `total_seconds()`.

```
from datetime import datetime


fecha1 = datetime(2025, 10, 1, 12, 0)
fecha2 = datetime(2025, 10, 2, 14, 30)
segundos_transcurridos = (fecha2 - fecha1).total_seconds()
print("Segundos transcurridos:", segundos_transcurridos)
```

Ejercicio 50

Enunciado: Obtén la fecha de inicio de la semana actual (lunes).
Explicación: Restamos los días desde el lunes usando `weekday()`.

```
from datetime import datetime, timedelta


hoy = datetime.now()
inicio_semana = hoy - timedelta(days=hoy.weekday())
print("Fecha del inicio de la semana (lunes):",
inicio_semana.date())
```

Ejercicio 51

Enunciado: Obtén la fecha de final de la semana actual (domingo).
Explicación: Sumamos los días restantes hasta el domingo usando `weekday()`.

```
from datetime import datetime, timedelta


hoy = datetime.now()
fin_semana = hoy + timedelta(days=(6 - hoy.weekday()))
print("Fecha del fin de semana (domingo):", fin_semana.date())
```

Ejercicio 52

Enunciado: Muestra la fecha del primer día del mes actual.
Explicación: Creamos un objeto `datetime` con día 1 y mes actual.

```
from datetime import datetime


hoy = datetime.now()
primer_dia_mes = datetime(hoy.year, hoy.month, 1)
print("Primer día del mes:", primer_dia_mes.date())
```

Ejercicio 53

Enunciado: Muestra la fecha del último día del mes actual.
Explicación: Usamos `calendar.monthrange` para obtener el número de días del mes.

```
from datetime import datetime
import calendar


hoy = datetime.now()
ultimo_dia = calendar.monthrange(hoy.year, hoy.month)[1]
fecha_ultimo_dia = datetime(hoy.year, hoy.month, ultimo_dia)
print("Último día del mes:", fecha_ultimo_dia.date())
```

Ejercicio 54

Enunciado: Calcula cuántos días faltan para el próximo viernes.
Explicación: Calculamos la diferencia hasta el viernes (weekday=4).

```
from datetime import datetime, timedelta

hoy = datetime.now()
dias_para_viernes = (4 - hoy.weekday() + 7) % 7
if dias_para_viernes == 0:
    dias_para_viernes = 7
proximo_viernes = hoy + timedelta(days=dias_para_viernes)
print("Días para el próximo viernes:", dias_para_viernes)
print("Fecha del próximo viernes:", proximo_viernes.date())
```

Ejercicio 55

Enunciado: Calcula la fecha 100 días antes de hoy.
Explicación: Restamos `timedelta(days=100)` a la fecha actual.

```
from datetime import datetime, timedelta

hoy = datetime.now()
fecha_100_dias_atras = hoy - timedelta(days=100)
print("Fecha 100 días antes de hoy:", fecha_100_dias_atras.date())
```

Ejercicio 56

Enunciado: Calcula la fecha 200 días después de hoy.
Explicación: Sumamos `timedelta(days=200)` a la fecha actual.

```
from datetime import datetime, timedelta

hoy = datetime.now()
fecha_200_dias_despues = hoy + timedelta(days=200)
print("Fecha 200 días después de hoy:",
fecha_200_dias_despues.date())
```

Ejercicio 57

Enunciado: Calcula cuántos días tiene el mes de abril de un año específico.
Explicación: Usamos `calendar.monthrange` para obtener la cantidad de días.

```
import calendar


año = 2025
dias_abril = calendar.monthrange(año, 4)[1]
print(f"Abril de {año} tiene {dias_abril} días")
```

Ejercicio 58

Enunciado: Muestra la fecha de hace 6 meses desde hoy.
Explicación: Restamos aproximadamente 182 días usando `timedelta`.

```
from datetime import datetime, timedelta


hoy = datetime.now()
fecha_6_meses_atras = hoy - timedelta(days=182)
print("Fecha de hace 6 meses:", fecha_6_meses_atras.date())
```

Ejercicio 59

Enunciado: Muestra la fecha dentro de 3 meses desde hoy.
Explicación: Sumamos aproximadamente 90 días usando `timedelta`.

```
from datetime import datetime, timedelta


hoy = datetime.now()
fecha_3_meses_despues = hoy + timedelta(days=90)
print("Fecha dentro de 3 meses:", fecha_3_meses_despues.date())
```

Ejercicio 60

Enunciado: Muestra el tiempo transcurrido hoy en horas y minutos.
Explicación: Calculamos la diferencia entre la hora actual y medianoche y convertimos a horas y minutos.

```
from datetime import datetime, time


ahora = datetime.now()
medianoche = datetime.combine(ahora.date(), time())
diferencia = ahora - medianoche
horas = diferencia.seconds // 3600
minutos = (diferencia.seconds % 3600) // 60
print(f"Tiempo transcurrido hoy: {horas} horas y {minutos} minutos")
```

Ejercicio 61

Enunciado: Calcula cuántos días tiene el mes de febrero de un año específico.
Explicación: Usamos `calendar.monthrange` para obtener el número de días de febrero considerando si es bisiesto o no.

```
import calendar


año = 2024
dias_febrero = calendar.monthrange(año, 2)[1]
print(f"Febrero de {año} tiene {dias_febrero} días")
```

Ejercicio 62

Enunciado: Muestra la fecha y hora actual en formato UTC.
Explicación: `datetime.utcnow()` devuelve la fecha y hora en tiempo universal coordinado.

```
from datetime import datetime


utc_actual = datetime.utcnow()
print("Fecha y hora actual UTC:", utc_actual)
```

Ejercicio 63

Enunciado: Calcula la fecha 10 días antes de tu cumpleaños.
Explicación: Restamos `timedelta(days=10)` a la fecha de cumpleaños.

```
from datetime import datetime, timedelta


cumpleaños = datetime(2025, 12, 15)
fecha_antes = cumpleaños - timedelta(days=10)
print("Fecha 10 días antes del cumpleaños:", fecha_antes.date())
```

Ejercicio 64

Enunciado: Calcula la fecha 15 días después de tu cumpleaños.
Explicación: Sumamos `timedelta(days=15)` a la fecha de cumpleaños.

```
from datetime import datetime, timedelta


cumpleaños = datetime(2025, 12, 15)
fecha_despues = cumpleaños + timedelta(days=15)
print("Fecha 15 días después del cumpleaños:",
fecha_despues.date())
```

Ejercicio 65

Enunciado: Muestra la fecha y hora actual en formato legible largo.
Explicación: `strftime` permite personalizar la salida incluyendo día de la semana, mes y hora.

```
from datetime import datetime


ahora = datetime.now()
fecha_hora_larga = ahora.strftime("%A, %d de %B de %Y %H:%M:%S")
print("Fecha y hora larga:", fecha_hora_larga)
```

Ejercicio 66

Enunciado: Calcula cuántos días han pasado desde el inicio del mes.
Explicación: Restamos la fecha actual con el primer día del mes.

```
from datetime import datetime


hoy = datetime.now()
primer_dia_mes = datetime(hoy.year, hoy.month, 1)
dias_transcurridos = (hoy - primer_dia_mes).days
print("Días desde inicio del mes:", dias_transcurridos)
```

Ejercicio 67

Enunciado: Calcula cuántos días faltan hasta fin de mes.
Explicación: Restamos la fecha actual con el último día del mes usando `calendar.monthrange`.

```
from datetime import datetime
import calendar


hoy = datetime.now()
ultimo_dia = calendar.monthrange(hoy.year, hoy.month)[1]
fin_mes = datetime(hoy.year, hoy.month, ultimo_dia)
faltan_dias = (fin_mes - hoy).days
print("Días para fin de mes:", faltan_dias)
```

Ejercicio 68

Enunciado: Calcula cuántos días tiene el año actual.
Explicación: Usamos `calendar.isleap` para determinar si el año es bisiesto y definir 365 o 366 días.

```
import calendar
from datetime import datetime


año_actual = datetime.now().year
dias_año = 366 if calendar.isleap(año_actual) else 365
print("Días del año actual:", dias_año)
```

Ejercicio 69

Enunciado: Calcula cuántos minutos han pasado desde medianoche.
Explicación: Restamos la hora actual con la medianoche y convertimos los segundos a minutos.

```
from datetime import datetime, time

ahora = datetime.now()
medianoche = datetime.combine(ahora.date(), time())
diferencia = ahora - medianoche
minutos = diferencia.seconds // 60
print("Minutos desde medianoche:", minutos)
```

Ejercicio 70

Enunciado: Muestra la fecha y hora actuales en formato de 24 horas y 12 horas juntos.
Explicación: Usamos `strftime` para mostrar la hora en ambos formatos.

```
from datetime import datetime

ahora = datetime.now()
hora_24 = ahora.strftime("%H:%M:%S")
hora_12 = ahora.strftime("%I:%M:%S %p")
print("Hora 24h:", hora_24)
print("Hora 12h:", hora_12)
```

Ejercicio 71

Enunciado: Muestra cuántos días faltan para la próxima Navidad.
Explicación: Creamos un objeto `datetime` con la fecha de Navidad y calculamos la diferencia.

```
from datetime import datetime

hoy = datetime.now()
navidad = datetime(hoy.year, 12, 25)
if navidad < hoy:
    navidad = datetime(hoy.year + 1, 12, 25)
faltan_dias = (navidad - hoy).days
print("Días para la próxima Navidad:", faltan_dias)
```

Ejercicio 72

Enunciado: Muestra la fecha de hace 7 días.
Explicación: Restamos 7 días a la fecha actual usando `timedelta`.

```
from datetime import datetime, timedelta


hoy = datetime.now()
hace_7_dias = hoy - timedelta(days=7)
print("Fecha hace 7 días:", hace_7_dias.date())
```

Ejercicio 73

Enunciado: Muestra la fecha de dentro de 7 días.
Explicación: Sumamos 7 días a la fecha actual usando `timedelta`.

```
from datetime import datetime, timedelta


hoy = datetime.now()
dentro_7_dias = hoy + timedelta(days=7)
print("Fecha dentro de 7 días:", dentro_7_dias.date())
```

Ejercicio 74

Enunciado: Calcula la cantidad de semanas transcurridas en el año actual.
Explicación: `isocalendar()[1]` devuelve la semana ISO del año.

```
from datetime import datetime


hoy = datetime.now()
semana_actual = hoy.isocalendar()[1]
print("Semana actual del año:", semana_actual)
```

Ejercicio 75

Enunciado: Calcula la fecha de hace 1 año desde hoy.
Explicación: Restamos aproximadamente 365 días usando `timedelta`.

```
from datetime import datetime, timedelta


hoy = datetime.now()
fecha_hace_un_año = hoy - timedelta(days=365)
print("Fecha de hace 1 año:", fecha_hace_un_año.date())
```

Ejercicio 76

Enunciado: Calcula la fecha de dentro de 1 año desde hoy.
Explicación: Sumamos aproximadamente 365 días usando `timedelta`.

```
from datetime import datetime, timedelta


hoy = datetime.now()
fecha_dentro_un_año = hoy + timedelta(days=365)
print("Fecha dentro de 1 año:", fecha_dentro_un_año.date())
```

Ejercicio 77

Enunciado: Muestra el número del trimestre actual del año.
Explicación: Dividimos el mes entre 3 y usamos `ceil` para obtener el trimestre.

```
from datetime import datetime
import math


hoy = datetime.now()
trimestre = math.ceil(hoy.month / 3)
print("Trimestre actual:", trimestre)
```

Ejercicio 78

Enunciado: Muestra la fecha y hora actual con zona horaria local.
Explicación: Usamos `datetime.now()` con `astimezone()` para obtener la zona local.

```
from datetime import datetime


ahora = datetime.now().astimezone()
print("Fecha y hora con zona horaria local:", ahora)
```

Ejercicio 79

Enunciado: Calcula cuántos días hay entre dos fechas dadas.
Explicación: Restamos dos objetos `datetime` y accedemos a `.days`.

```
from datetime import datetime


fecha1 = datetime(2025, 1, 1)
fecha2 = datetime(2025, 10, 26)
dias_entre = (fecha2 - fecha1).days
print("Días entre las dos fechas:", dias_entre)
```

Ejercicio 80

Enunciado: Muestra la fecha actual en formato "DD de mes de AAAA".
Explicación: `strftime("%d de %B de %Y")` permite mostrar la fecha de forma descriptiva.

```
from datetime import datetime


hoy = datetime.now()
fecha_descriptiva = hoy.strftime("%d de %B de %Y")
print("Fecha descriptiva:", fecha_descriptiva)
```

Ejercicio 81

Enunciado: Muestra el primer día del año actual.
Explicación: Creamos un objeto `datetime` con día 1 y mes 1 del año actual.

```
from datetime import datetime


hoy = datetime.now()
primer_dia_año = datetime(hoy.year, 1, 1)
print("Primer día del año:", primer_dia_año.date())
```

Ejercicio 82

Enunciado: Muestra el último día del año actual.
Explicación: Creamos un objeto `datetime` con día 31 y mes 12 del año actual.

```
from datetime import datetime


hoy = datetime.now()
ultimo_dia_año = datetime(hoy.year, 12, 31)
print("Último día del año:", ultimo_dia_año.date())
```

Ejercicio 83

Enunciado: Muestra la fecha de hace 30 minutos desde ahora.
Explicación: Restamos `timedelta(minutes=30)` a la fecha y hora actual.

```
from datetime import datetime, timedelta


ahora = datetime.now()
hace_30_min = ahora - timedelta(minutes=30)
print("Fecha y hora hace 30 minutos:", hace_30_min)
```

Ejercicio 84

Enunciado: Muestra la fecha de dentro de 30 minutos desde ahora.
Explicación: Sumamos `timedelta(minutes=30)` a la fecha y hora actual.

```
from datetime import datetime, timedelta


ahora = datetime.now()
dentro_30_min = ahora + timedelta(minutes=30)
print("Fecha y hora dentro de 30 minutos:", dentro_30_min)
```

Ejercicio 85

Enunciado: Calcula la cantidad de días entre hoy y el inicio del año.
Explicación: Restamos la fecha actual con el primer día del año y usamos `.days`.

```
from datetime import datetime


hoy = datetime.now()
inicio_año = datetime(hoy.year, 1, 1)
dias_transcurridos = (hoy - inicio_año).days
print("Días transcurridos desde inicio del año:",
dias_transcurridos)
```

Ejercicio 86

Enunciado: Calcula cuántos días faltan para el final del año.
Explicación: Restamos el último día del año con la fecha actual.

```
from datetime import datetime


hoy = datetime.now()
fin_año = datetime(hoy.year, 12, 31)
faltan_dias = (fin_año - hoy).days
print("Días para fin de año:", faltan_dias)
```

Ejercicio 87

Enunciado: Muestra la fecha actual con hora, minuto y segundo en formato personalizado.
Explicación: `strftime("%d/%m/%Y %H:%M:%S")` permite un formato detallado de fecha y hora.

```
from datetime import datetime


ahora = datetime.now()
fecha_hora = ahora.strftime("%d/%m/%Y %H:%M:%S")
print("Fecha y hora personalizada:", fecha_hora)
```

Ejercicio 88

Enunciado: Calcula la fecha exacta hace 90 días desde hoy.
Explicación: Restamos `timedelta(days=90)` a la fecha actual.

```
from datetime import datetime, timedelta


hoy = datetime.now()
fecha_90_dias_atras = hoy - timedelta(days=90)
print("Fecha hace 90 días:", fecha_90_dias_atras.date())
```

Ejercicio 89

Enunciado: Calcula la fecha exacta dentro de 90 días desde hoy.
Explicación: Sumamos `timedelta(days=90)` a la fecha actual.

```
from datetime import datetime, timedelta


hoy = datetime.now()
fecha_90_dias_despues = hoy + timedelta(days=90)
print("Fecha dentro de 90 días:", fecha_90_dias_despues.date())
```

Ejercicio 90

Enunciado: Muestra la fecha y hora actuales con microsegundos.
Explicación: `datetime.now()` incluye microsegundos que podemos mostrar directamente.

```
from datetime import datetime


ahora = datetime.now()
print("Fecha y hora con microsegundos:", ahora)
```

Ejercicio 91

Enunciado: Muestra la fecha y hora actuales en formato ISO 8601 con zona horaria.
Explicación: `astimezone().isoformat()` permite obtener la fecha y hora con zona local en formato estándar.

```
from datetime import datetime


ahora = datetime.now().astimezone()
print("Fecha y hora ISO con zona horaria:", ahora.isoformat())
```

Ejercicio 92

Enunciado: Calcula la fecha de hace 6 días y 4 horas.
Explicación: `timedelta(days=6, hours=4)` permite restar días y horas combinados.

```
from datetime import datetime, timedelta


ahora = datetime.now()
fecha_modificada = ahora - timedelta(days=6, hours=4)
print("Fecha hace 6 días y 4 horas:", fecha_modificada)
```

Ejercicio 93

Enunciado: Calcula la fecha de dentro de 10 días y 5 horas.
Explicación: `timedelta(days=10, hours=5)` permite sumar días y horas combinados.

```
from datetime import datetime, timedelta


ahora = datetime.now()
fecha_modificada = ahora + timedelta(days=10, hours=5)
print("Fecha dentro de 10 días y 5 horas:", fecha_modificada)
```

Ejercicio 94

Enunciado: Muestra la fecha y hora actual solo con horas y minutos.
Explicación: `strftime("%H:%M")` permite mostrar únicamente horas y minutos.

```
from datetime import datetime


ahora = datetime.now()
hora_minutos = ahora.strftime("%H:%M")
print("Hora y minutos actuales:", hora_minutos)
```

Ejercicio 95

Enunciado: Muestra el día de la semana de una fecha específica.
Explicación: `strftime("%A")` devuelve el nombre completo del día de la semana.

```
from datetime import datetime


fecha = datetime(2025, 12, 31)
dia_semana = fecha.strftime("%A")
print("Día de la semana:", dia_semana)
```

Ejercicio 96

Enunciado: Muestra el nombre abreviado del mes de una fecha específica.
Explicación: `strftime("%b")` devuelve el nombre del mes abreviado.

```
from datetime import datetime


fecha = datetime(2025, 12, 31)
mes_abreviado = fecha.strftime("%b")
print("Mes abreviado:", mes_abreviado)
```

Ejercicio 97

Enunciado: Calcula la diferencia en horas entre dos fechas y horas.
Explicación: Restamos dos objetos `datetime` y usamos `.total_seconds()/3600` para obtener horas.

```
from datetime import datetime


fecha1 = datetime(2025, 10, 26, 12, 0)
fecha2 = datetime(2025, 10, 27, 18, 0)
horas_diferencia = (fecha2 - fecha1).total_seconds() / 3600
print("Diferencia en horas:", horas_diferencia)
```

Ejercicio 98

Enunciado: Calcula la diferencia en minutos entre dos fechas y horas.
Explicación: Restamos dos objetos `datetime` y usamos `.total_seconds()/60` para obtener minutos.

```
from datetime import datetime


fecha1 = datetime(2025, 10, 26, 12, 0)
fecha2 = datetime(2025, 10, 26, 14, 30)
minutos_diferencia = (fecha2 - fecha1).total_seconds() / 60
print("Diferencia en minutos:", minutos_diferencia)
```

Ejercicio 99

Enunciado: Muestra la fecha actual en formato legible con día ordinal (1st, 2nd, 3rd…).
Explicación: Usamos una función auxiliar para el ordinal y formateamos la fecha.

```
from datetime import datetime


def ordinal(n):
    return "%d%s" % (n,"tsnrhtdd"[(n//10%10!=1)*(n%10<4)*n%10::4])


hoy = datetime.now()
print(f"Fecha con ordinal: {ordinal(hoy.day)} {hoy.strftime('%B %Y')}")
```

Ejercicio 100

Enunciado: Muestra la fecha y hora actual en formato completo con nombre de día y mes.
Explicación: `strftime("%A, %d de %B de %Y %H:%M:%S")` permite una salida completa y legible.

```
from datetime import datetime


ahora = datetime.now()
fecha_completa = ahora.strftime("%A, %d de %B de %Y %H:%M:%S")
print("Fecha y hora completa:", fecha_completa)
```

Ejercicios python formato

Ejercicio 1

Enunciado: Imprime una cadena de texto simple.
Explicación: Usamos `print()` para mostrar texto en la consola.

```
print("¡Hola, mundo!")
```

Ejercicio 2

Enunciado: Imprime varias cadenas en la misma línea usando comas.
Explicación: La función `print()` permite separar argumentos con comas.

```
print("Hola,", "¿cómo", "estás?")
```

Ejercicio 3

Enunciado: Imprime varias cadenas sin espacio usando `sep=""`.
Explicación: El parámetro `sep` define el separador entre los elementos.

```
print("Python", "es", "genial", sep="-")
```

Ejercicio 4

Enunciado: Imprime sin salto de línea usando `end=""`.
Explicación: Por defecto, `print()` añade un salto de línea. `end=""` evita esto.

```
print("Hola,", end=" ")
print("mundo!")
```

Ejercicio 5

Enunciado: Usa formato con f-strings para mostrar variables.
Explicación: `f"{variable}"` permite incluir variables dentro de cadenas de manera sencilla.

```
nombre = "Ana"
edad = 25
print(f"Hola, mi nombre es {nombre} y tengo {edad} años.")
```

Ejercicio 6

Enunciado: Muestra un número con dos decimales usando f-string.
Explicación: `{variable:.2f}` permite formatear flotantes a 2 decimales.

```
precio = 19.9876
print(f"El precio es ${precio:.2f}")
```

Ejercicio 7

Enunciado: Alinea texto a la derecha usando f-string y ancho fijo.
Explicación: `{variable:>10}` alinea el texto a la derecha en 10 espacios.

```
nombre = "Ana"
print(f"{nombre:>10}")
```

Ejercicio 8

Enunciado: Alinea texto a la izquierda usando f-string y ancho fijo.
Explicación: `{variable:<10}` alinea el texto a la izquierda en 10 espacios.

```
nombre = "Ana"
print(f"{nombre:<10}fin")
```

Ejercicio 9

Enunciado: Centra texto usando f-string y ancho fijo.
Explicación: `{variable:^10}` centra el texto en 10 espacios.

```
nombre = "Ana"
print(f"{nombre:^10}fin")
```

Ejercicio 10

Enunciado: Muestra un número con relleno de ceros a la izquierda.
Explicación: `{variable:0>5}` completa con ceros hasta 5 dígitos.

```
numero = 42
print(f"{numero:0>5}")
```

Ejercicio 11

Enunciado: Muestra un número con signo positivo y negativo usando f-string.
Explicación: `{variable:+}` muestra el signo del número.

```
num1 = 42
num2 = -7
print(f"Num1: {num1:+}, Num2: {num2:+}")
```

Ejercicio 12

Enunciado: Muestra un número en notación científica.
Explicación: `{variable:e}` permite mostrar un número en notación científica.

```
numero = 1234567
print(f"Notación científica: {numero:e}")
```

Ejercicio 13

Enunciado: Muestra un número en porcentaje.
Explicación: `{variable:.2%}` multiplica por 100 y agrega el símbolo %.

```
progreso = 0.857
print(f"Progreso: {progreso:.2%}")
```

Ejercicio 14

Enunciado: Muestra un número con separador de miles.
Explicación: `{variable:,}` añade comas para separar miles.

```
numero = 1234567
print(f"Número con separador de miles: {numero:,}")
```

Ejercicio 15

Enunciado: Muestra un número con relleno de espacios a la izquierda.
Explicación: `{variable:>10}` alinea a la derecha en 10 espacios.

```
numero = 42
print(f"{numero:>10} fin")
```

Ejercicio 16

Enunciado: Muestra un número con relleno de espacios a la izquierda, y decimales.
Explicación: `{variable:>10.2f}` alinea a la derecha con 10 espacios y 2 decimales.

```
numero = 3.14159
print(f"{numero:>10.2f}")
```

Ejercicio 17

Enunciado: Alinea texto a la izquierda y rellena con puntos.
Explicación: `{variable:.<10}` alinea a la izquierda y rellena el resto con el carácter `.`.

```
texto = "Hola"
print(f"{texto:.<10}fin")
```

Ejercicio 18

Enunciado: Alinea texto a la derecha y rellena con asteriscos.
Explicación: `{variable:*>10}` alinea a la derecha y completa con `*`.

```
texto = "Hola"
print(f"{texto:*>10}fin")
```

Ejercicio 19

Enunciado: Centra texto y rellena con guiones.
Explicación: `{variable:-^10}` centra el texto y completa con `-`.

```
texto = "Hi"
print(f"{texto:-^10}fin")
```

Ejercicio 20

Enunciado: Combina varios formatos: alinea, rellena y muestra decimal con 2 cifras.
Explicación: `{variable:*>10.2f}` combina alineación, relleno y formato decimal.

```
numero = 7.5
print(f"{numero:*>10.2f}")
```

Ejercicio 21

Enunciado: Muestra texto con comillas dentro de la cadena.
Explicación: Se pueden usar comillas simples dentro de comillas dobles o viceversa.

```
print("Ella dijo: 'Hola, ¿cómo estás?'")
```

Ejercicio 22

Enunciado: Muestra texto con salto de línea.
Explicación: `\n` permite insertar un salto de línea dentro de la cadena.

```
print("Primera línea\nSegunda línea")
```

Ejercicio 23

Enunciado: Muestra texto con tabulación.
Explicación: `\t` agrega un tabulador en la cadena.

```
print("Columna1\tColumna2\tColumna3")
```

Ejercicio 24

Enunciado: Escapa caracteres especiales dentro de una cadena.
Explicación: `\\` permite mostrar la barra invertida literal.

```
print("Ruta de carpeta: C:\\Users\\Ana")
```

Ejercicio 25

Enunciado: Usa triple comilla para texto multilínea.
Explicación: `""" """` permite escribir varias líneas sin `\n`.

```
print("""Esto es
un texto
multilínea""")
```

Ejercicio 26

Enunciado: Inserta variables en texto usando `str.format()`.
Explicación: `{}` en el texto se reemplaza con los valores de `.format()`.

```
nombre = "Ana"
edad = 25
print("Hola, mi nombre es {} y tengo {} años.".format(nombre, edad))
```

Ejercicio 27

Enunciado: Ordena los valores en `str.format()` según índice.

Explicación: `{1}` y `{0}` permiten reordenar los valores.

```
a = "mundo"
b = "Hola"
print("{1}, {0}!".format(a, b))
```

Ejercicio 28

Enunciado: Formatea un número con dos decimales usando `str.format()`.

Explicación: `:.2f` dentro de `{}` indica dos decimales.

```
precio = 19.9876
print("El precio es ${:.2f}".format(precio))
```

Ejercicio 29

Enunciado: Alinea texto a la derecha con `str.format()`.

Explicación: `{:>10}` alinea el texto a la derecha con ancho 10.

```
nombre = "Ana"
print("{:>10}".format(nombre))
```

Ejercicio 30

Enunciado: Alinea texto a la izquierda con `str.format()`.

Explicación: `{: <10}` alinea el texto a la izquierda con ancho 10.

```
nombre = "Ana"
print("{:<10}fin".format(nombre))
```

Ejercicio 31

Enunciado: Centra texto con `str.format()` y ancho fijo.

Explicación: `{:^10}` centra el texto dentro de 10 espacios.

```
texto = "Hola"
print("{:^10}fin".format(texto))
```

Ejercicio 32

Enunciado: Rellena con asteriscos al alinear a la izquierda.
Explicación: `:*<10` rellena los espacios restantes con `*`.

```
texto = "Hola"
print("{:*<10}fin".format(texto))
```

Ejercicio 33

Enunciado: Rellena con guiones al alinear a la derecha.
Explicación: `:->10` rellena los espacios a la izquierda con `-`.

```
texto = "Hola"
print("{:->10}fin".format(texto))
```

Ejercicio 34

Enunciado: Rellena con puntos al centrar texto.
Explicación: `:.^10` centra el texto y rellena los espacios con `.`

```
texto = "Hi"
print("{:.^10}fin".format(texto))
```

Ejercicio 35

Enunciado: Muestra un número con signo positivo y negativo usando `str.format()`.
Explicación: `{:+}` muestra el signo del número.

```
num1 = 42
num2 = -7
print("Num1: {:+}, Num2: {:+}".format(num1, num2))
```

Ejercicio 36

Enunciado: Muestra un número en notación científica usando `str.format()`.
Explicación: `{:e}` permite mostrar números en notación científica.

```
numero = 1234567
print("Notación científica: {:e}".format(numero))
```

Ejercicio 37

Enunciado: Muestra un número en porcentaje usando `str.format()`.
Explicación: `:.2%` multiplica por 100 y agrega %.

```
progreso = 0.857
print("Progreso: {:.2%}".format(progreso))
```

Ejercicio 38

Enunciado: Muestra un número con separador de miles usando `str.format()`.
Explicación: `{: ,}` agrega comas para separar miles.

```
numero = 1234567
print("Número con separador de miles: {:,}".format(numero))
```

Ejercicio 39

Enunciado: Combina alineación a la derecha con relleno de ceros y decimales.
Explicación: `{:0>10.2f}` completa con ceros, ancho 10 y 2 decimales.

```
numero = 7.5
print("{:0>10.2f}".format(numero))
```

Ejercicio 40

Enunciado: Muestra un número centrado con relleno de guiones y decimales.
Explicación: `:-^10.2f` centra el número, rellena con `-` y muestra 2 decimales.

```
numero = 3.14
print("{:-^10.2f}".format(numero))
```

Si quieres, puedo continuar con el **Bloque 5 (ejercicios 41 al 50)** para seguir avanzando hacia los 100 programas de formato en Python.

¿Deseas que continúe con el **Bloque 5**?

¡Perfecto! 🌟 Aquí tienes el **Bloque 5 de 10 programas (41 al 50)** sobre **formato en Python**, nivel fácil, con enunciado, explicación y código:

Ejercicio 41

Enunciado: Muestra texto con comillas dobles dentro de comillas simples.
Explicación: Se pueden combinar comillas simples y dobles para incluir texto con comillas.

```
print('Ella dijo: "Hola, ¿cómo estás?"')
```

Ejercicio 42

Enunciado: Imprime varias líneas usando salto de línea `\n`.
Explicación: `\n` permite insertar saltos de línea dentro de la cadena.

```
print("Línea 1\nLínea 2\nLínea 3")
```

Ejercicio 43

Enunciado: Imprime texto con tabulaciones usando `\t`.
Explicación: `\t` crea espacios de tabulación entre columnas de texto.

```
print("Nombre\tEdad\tCiudad")
print("Ana\t25\tZaragoza")
```

Ejercicio 44

Enunciado: Escapa caracteres especiales dentro de una cadena.
Explicación: `\\` permite mostrar una barra invertida literal.

```
print("Ruta de carpeta: C:\\Users\\Ana\\Documentos")
```

Ejercicio 45

Enunciado: Usa triple comilla para escribir texto multilínea.
Explicación: `""" """` permite escribir varias líneas sin necesidad de `\n`.

```
print("""Este es un texto
que ocupa varias
líneas""")
```

Ejercicio 46

Enunciado: Inserta variables en una cadena usando f-string.
Explicación: `f"{variable}"` permite incluir variables de forma sencilla dentro de la cadena.

```
nombre = "Ana"
edad = 25
print(f"Hola, mi nombre es {nombre} y tengo {edad} años.")
```

Ejercicio 47

Enunciado: Inserta variables en una cadena usando `str.format()`.
Explicación: `{}` se reemplaza por los valores pasados a `.format()`.

```
nombre = "Ana"
edad = 25
print("Hola, mi nombre es {} y tengo {} años.".format(nombre, edad))
```

Ejercicio 48

Enunciado: Muestra un número con 2 decimales usando f-string.
Explicación: `{variable:.2f}` formatea el número a 2 decimales.

```
precio = 19.9876
print(f"El precio es ${precio:.2f}")
```

Ejercicio 49

Enunciado: Muestra un número con 2 decimales usando `str.format()`.
Explicación: `{: .2f}` formatea el número a 2 decimales.

```
precio = 19.9876
print("El precio es ${:.2f}".format(precio))
```

Ejercicio 50

Enunciado: Muestra un número con separador de miles usando f-string.
Explicación: `{variable:,}` agrega comas para separar miles.

```
numero = 1234567
print(f"Número con separador de miles: {numero:,}")
```

Si quieres, puedo continuar con el **Bloque 6 (ejercicios 51 al 60)** para seguir avanzando hacia los 100 programas de formato en Python.

¿Deseas que continúe con el **Bloque 6**?

¡Perfecto! 🌟 Aquí tienes el **Bloque 6 de 10 programas (51 al 60)** sobre **formato en Python**, nivel fácil, con enunciado, explicación y código:

Ejercicio 51

Enunciado: Muestra un número con signo usando f-string.
Explicación: `{variable:+}` muestra el signo positivo o negativo del número.

```
num1 = 42
num2 = -7
print(f"Num1: {num1:+}, Num2: {num2:+}")
```

Ejercicio 52

Enunciado: Muestra un número en notación científica usando f-string.
Explicación: `{variable:e}` muestra un número en notación científica.

```
numero = 1234567
print(f"Notación científica: {numero:e}")
```

Ejercicio 53

Enunciado: Muestra un número en porcentaje usando f-string.
Explicación: `{variable:.2%}` multiplica por 100 y agrega el símbolo %.

```
progreso = 0.857
print(f"Progreso: {progreso:.2%}")
```

Ejercicio 54

Enunciado: Muestra un número con separador de miles usando f-string.
Explicación: `{variable:,}` agrega comas para separar miles.

```
numero = 1234567
print(f"Número con separador de miles: {numero:,}")
```

Ejercicio 55

Enunciado: Alinea texto a la derecha y rellena con espacios.
Explicación: `{variable:>10}` alinea a la derecha en 10 espacios.

```
texto = "Hola"
print(f"{texto:>10}fin")
```

Ejercicio 56

Enunciado: Alinea texto a la izquierda y rellena con espacios.
Explicación: `{variable:<10}` alinea a la izquierda en 10 espacios.

```
texto = "Hola"
print(f"{texto:<10}fin")
```

Ejercicio 57

Enunciado: Centra texto con relleno de puntos.
Explicación: `{variable:.^10}` centra el texto y rellena con `.`

```
texto = "Hi"
print(f"{texto:.^10}fin")
```

Ejercicio 58

Enunciado: Rellena texto alineado a la derecha con asteriscos.
Explicación: `{variable:*>10}` alinea a la derecha y completa con `*`.

```
texto = "Hola"
print(f"{texto:*>10}fin")
```

Ejercicio 59

Enunciado: Rellena texto alineado a la izquierda con guiones.
Explicación: `{variable:-<10}` alinea a la izquierda y completa con `-`.

```
texto = "Hola"
print(f"{texto:-<10}fin")
```

Ejercicio 60

Enunciado: Combina alineación centrada, relleno y formato decimal.
Explicación: `{:*^10.2f}` centra el número, rellena con `*` y muestra 2 decimales.

```
numero = 3.14159
print(f"{numero:*^10.2f}")
```

Ejercicio 61

Enunciado: Muestra texto con comillas simples dentro de comillas dobles.
Explicación: Se puede incluir comillas simples dentro de comillas dobles sin necesidad de escape.

```
print("Ella dijo: 'Python es genial!'")
```

Ejercicio 62

Enunciado: Muestra texto con comillas dobles dentro de comillas simples.
Explicación: Se puede incluir comillas dobles dentro de comillas simples sin necesidad de escape.

```
print('Él dijo: "Me gusta programar"')
```

Ejercicio 63

Enunciado: Inserta variables en texto usando f-string y combinación de texto fijo.
Explicación: Se pueden combinar variables y texto estático fácilmente.

```
nombre = "Ana"
edad = 25
print(f"{nombre} tiene {edad} años y está aprendiendo Python.")
```

Ejercicio 64

Enunciado: Inserta variables en texto usando `str.format()` y texto fijo.
Explicación: `.format()` permite combinar texto y variables de manera clara.

```
nombre = "Ana"
edad = 25
print("{} tiene {} años y está aprendiendo Python.".format(nombre, edad))
```

Ejercicio 65

Enunciado: Muestra número con dos decimales usando f-string.
Explicación: `{variable:.2f}` formatea números flotantes con dos decimales.

```
precio = 19.9876
print(f"El precio final es ${precio:.2f}")
```

Ejercicio 66

Enunciado: Muestra número con dos decimales usando `str.format()`.
Explicación: `{: .2f}` dentro de `.format()` permite controlar decimales.

```
precio = 19.9876
print("El precio final es ${:.2f}".format(precio))
```

Ejercicio 67

Enunciado: Alinea texto a la derecha con relleno de ceros usando f-string.
Explicación: `{variable:0>10}` alinea a la derecha y completa con ceros hasta 10 caracteres.

```
numero = 42
print(f"{numero:0>10}")
```

Ejercicio 68

Enunciado: Alinea texto a la izquierda con relleno de asteriscos usando f-string.
Explicación: `{variable:*<10}` alinea a la izquierda y completa con `*`.

```
texto = "Hola"
print(f"{texto:*<10}")
```

Ejercicio 69

Enunciado: Centra texto con relleno de guiones usando f-string.
Explicación: `{variable:-^10}` centra el texto y rellena con `-`.

```
texto = "Hi"
print(f"{texto:-^10}")
```

Ejercicio 70

Enunciado: Combina relleno, alineación y decimales en un número.
Explicación: `{:*^10.2f}` centra el número, rellena con `*` y muestra 2 decimales.

```
numero = 3.14159
print(f"{numero:*^10.2f}")
```

Ejercicio 71

Enunciado: Muestra un número con signo usando f-string y relleno.
Explicación: `{variable:+0>6}` muestra el signo y completa con ceros hasta 6 caracteres.

```
numero = 42
print(f"{numero:+0>6}")
```

Ejercicio 72

Enunciado: Muestra un número en notación científica con 3 decimales.
Explicación: `{variable:.3e}` formatea en notación científica con 3 decimales.

```
numero = 1234567
print(f"Notación científica: {numero:.3e}")
```

Ejercicio 73

Enunciado: Muestra un porcentaje con un decimal usando f-string.
Explicación: `{variable:.1%}` multiplica por 100, agrega % y muestra 1 decimal.

```
progreso = 0.857
print(f"Progreso: {progreso:.1%}")
```

Ejercicio 74

Enunciado: Muestra número con separador de miles y decimales.
Explicación: `{variable:,.2f}` combina coma para miles y 2 decimales.

```
numero = 1234567.891
print(f"Número formateado: {numero:,.2f}")
```

Ejercicio 75

Enunciado: Alinea texto a la derecha y rellena con puntos usando f-string.
Explicación: `{variable:.>10}` alinea a la derecha y completa con `.`

```
texto = "Hola"
print(f"{texto:.>10}fin")
```

Ejercicio 76

Enunciado: Alinea texto a la izquierda y rellena con guiones usando f-string.
Explicación: `{variable:-<10}` alinea a la izquierda y completa con `-`.

```
texto = "Hola"
print(f"{texto:-<10}fin")
```

Ejercicio 77

Enunciado: Centra texto con relleno de asteriscos usando f-string.
Explicación: `{variable:*^10}` centra el texto y completa con `*`.

```
texto = "Hi"
print(f"{texto:*^10}fin")
```

Ejercicio 78

Enunciado: Muestra un número con relleno de ceros a la izquierda y 2 decimales.
Explicación: `{:0>8.2f}` completa con ceros y muestra 2 decimales.

```
numero = 7.5
print(f"{numero:0>8.2f}")
```

Ejercicio 79

Enunciado: Muestra un número centrado con relleno de guiones y 2 decimales.
Explicación: `:-^10.2f` centra el número, rellena con `-` y muestra 2 decimales.

```
numero = 3.14159
print(f"{numero:-^10.2f}")
```

Ejercicio 80

Enunciado: Muestra un número con signo, relleno de asteriscos y decimales.
Explicación: `{:+*^10.2f}` muestra signo, centra, rellena con `*` y 2 decimales.

```
numero = 42.5
print(f"{numero:+*^10.2f}")
```

Ejercicio 81

Enunciado: Muestra un número en notación científica con signo.
Explicación: `{:+.2e}` muestra el signo y formatea el número en notación científica con 2 decimales.

```
numero = 12345
print(f"Número en notación científica con signo: {numero:+.2e}")
```

Ejercicio 82

Enunciado: Muestra un porcentaje con 3 decimales.
Explicación: `{:.3%}` multiplica por 100, agrega % y muestra 3 decimales.

```
progreso = 0.8573
print(f"Progreso: {progreso:.3%}")
```

Ejercicio 83

Enunciado: Muestra un número con separador de miles y signo.
Explicación: `{:+,}` combina signo y separador de miles.

```
numero = 1234567
print(f"Número con signo y separador de miles: {numero:+,}")
```

Ejercicio 84

Enunciado: Alinea texto a la derecha con relleno de ceros.
Explicación: `{variable:0>10}` completa con ceros hasta 10 caracteres.

```
texto = "42"
print(f"{texto:0>10}")
```

Ejercicio 85

Enunciado: Alinea texto a la izquierda con relleno de guiones.
Explicación: `{variable:-<10}` alinea a la izquierda y rellena con `-`.

```
texto = "Hola"
print(f"{texto:-<10}fin")
```

Ejercicio 86

Enunciado: Centra texto con relleno de puntos.
Explicación: `{variable:.^10}` centra el texto y completa con `.`

```
texto = "Hi"
print(f"{texto:.^10}")
```

Ejercicio 87

Enunciado: Muestra número con relleno de ceros y 3 decimales.
Explicación: `{:0>10.3f}` completa con ceros y muestra 3 decimales.

```
numero = 7.1234
print(f"{numero:0>10.3f}")
```

Ejercicio 88

Enunciado: Centra número con relleno de asteriscos y 2 decimales.
Explicación: `{:*^10.2f}` centra el número, rellena con `*` y muestra 2 decimales.

```
numero = 3.14
print(f"{numero:*^10.2f}")
```

Ejercicio 89

Enunciado: Muestra número con signo, relleno de guiones y 2 decimales.
Explicación: `{:+-^10.2f}` combina signo, centrado, relleno con `-` y 2 decimales.

```
numero = 42.5
print(f"{numero:+-^10.2f}")
```

Ejercicio 90

Enunciado: Combina signo, relleno de asteriscos, centrado y notación científica.
Explicación: `{:+*^12.2e}` muestra signo, centra, rellena con `*` y formatea en notación científica.

```
numero = 12345
print(f"{numero:+*^12.2e}")
```

Ejercicio 91

Enunciado: Muestra un número con separador de miles, signo y 2 decimales.
Explicación: `{:+,.2f}` combina signo, coma para miles y 2 decimales.

```
numero = 1234567.891
print(f"Número formateado: {numero:+,.2f}")
```

Ejercicio 92

Enunciado: Muestra un porcentaje centrado con relleno de guiones.
Explicación: `:-^10.1%` centra el porcentaje, rellena con `-` y muestra 1 decimal.

```
progreso = 0.857
print(f"{progreso:-^10.1%}"
```

Ejercicio 93

Enunciado: Muestra un número en notación científica con relleno de asteriscos y centrado.
Explicación: `:*^12.2e` centra el número, rellena con `*` y formatea en notación científica.

```
numero = 1234567
print(f"{numero:*^12.2e}")
```

Ejercicio 94

Enunciado: Muestra texto centrado con relleno de puntos y ancho 15.
Explicación: `{:.^15}` centra el texto y completa con `.` hasta 15 caracteres.

```
texto = "Python"
print(f"{texto:.^15}")
```

Ejercicio 95

Enunciado: Muestra texto alineado a la izquierda con relleno de guiones y ancho 12.
Explicación: `{-<12}` alinea a la izquierda y completa con `-`.

```
texto = "Hola"
print(f"{texto:-<12}fin")
```

Ejercicio 96

Enunciado: Muestra texto alineado a la derecha con relleno de asteriscos y ancho 12.
Explicación: `{:*>12}` alinea a la derecha y rellena con `*`.

```
texto = "Hola"
print(f"{texto:*>12}fin")
```

Ejercicio 97

Enunciado: Muestra un número con signo, centrado y relleno de guiones.
Explicación: `{:+-^10.2f}` combina signo, centrado, relleno con `-` y 2 decimales.

```
numero = 42.5
print(f"{numero:+-^10.2f}")
```

Ejercicio 98

Enunciado: Muestra un número en porcentaje con relleno de puntos y centrado.
Explicación: `{:.^12.1%}` centra el porcentaje, rellena con `.` y muestra 1 decimal.

```
progreso = 0.857
print(f"{progreso:.^12.1%}")
```

Ejercicio 99

Enunciado: Muestra un número en notación científica con signo y 3 decimales.
Explicación: `{:+.3e}` combina signo y notación científica con 3 decimales.

```
numero = 12345
print(f"{numero:+.3e}")
```

Ejercicio 100

Enunciado: Muestra un número con signo, relleno de asteriscos, centrado y 2 decimales.
Explicación: `{:+*^10.2f}` combina signo, centrado, relleno con `*` y 2 decimales.

```
numero = 42.75
print(f"{numero:+*^10.2f}")
```

Ejercicios python con archivos

1

Enunciado: Crear un archivo de texto llamado `hola.txt` y escribir "¡Hola, mundo!".

```
# ejercicio_01.py
with open("hola.txt", "w", encoding="utf-8") as f:
    f.write("¡Hola, mundo!")
print("Archivo 'hola.txt' creado.")
```

Explicación: Usa `open` en modo `"w"` para crear/escribir el archivo. `encoding="utf-8"` garantiza caracteres españoles.

2

Enunciado: Leer y mostrar el contenido de `hola.txt`.

```
# ejercicio_02.py
with open("hola.txt", "r", encoding="utf-8") as f:
    contenido = f.read()
print("Contenido de hola.txt:")
print(contenido)
```

Explicación: `open` en modo `"r"` lee todo el archivo; `read()` devuelve su contenido.

3

Enunciado: Añadir una línea nueva al final de `hola.txt`.

```
# ejercicio_03.py
with open("hola.txt", "a", encoding="utf-8") as f:
    f.write("\nEsta es una línea añadida.")
print("Línea añadida a 'hola.txt'.")
```

Explicación: Modo `"a"` (append) añade datos al final sin borrar el contenido previo.

4

Enunciado: Crear 5 archivos de texto numerados `fichero1.txt` ... `fichero5.txt`.

```
# ejercicio_04.py
for i in range(1, 6):
    with open(f"fichero{i}.txt", "w", encoding="utf-8") as f:
        f.write(f"Este es el fichero {i}\n")
print("5 archivos creados.")
```

Explicación: Bucle que crea varios archivos con nombres formateados.

5

Enunciado: Listar los archivos del directorio actual.

```
# ejercicio_05.py
import os
archivos = os.listdir(".")
print("Archivos y carpetas en el directorio actual:")
for a in archivos:
    print(a)
```

Explicación: `os.listdir(".")` devuelve nombres en el directorio actual.

6

Enunciado: Comprobar si existe `hola.txt`.

```
# ejercicio_06.py
from pathlib import Path
existe = Path("hola.txt").exists()
print("hola.txt existe?" , existe)
```

Explicación: `pathlib.Path.exists()` devuelve `True` si el archivo o carpeta existe.

7

Enunciado: Crear una carpeta llamada `mi_carpeta`.

```
# ejercicio_07.py
import os
os.makedirs("mi_carpeta", exist_ok=True)
print("Carpeta 'mi_carpeta' creada (o ya existía).")
```

Explicación: `os.makedirs` crea directorios, `exist_ok=True` evita error si ya existe.

8

Enunciado: Mover `fichero1.txt` a `mi_carpeta`.

```
# ejercicio_08.py
import shutil
shutil.move("fichero1.txt", "mi_carpeta/fichero1.txt")
print("fichero1.txt movido a mi_carpeta.")
```

Explicación: `shutil.move` mueve archivos (o los renombra).

9

Enunciado: Copiar `fichero2.txt` a `mi_carpeta`.

```
# ejercicio_09.py
import shutil
shutil.copy("fichero2.txt", "mi_carpeta/fichero2.txt")
print("fichero2.txt copiado a mi_carpeta.")
```

Explicación: `shutil.copy` copia el archivo (contenido + permisos básicos).

10

Enunciado: Eliminar `fichero3.txt`.

```
# ejercicio_10.py
import os
if os.path.exists("fichero3.txt"):
    os.remove("fichero3.txt")
    print("fichero3.txt eliminado.")
else:
    print("fichero3.txt no existe.")
```

Explicación: `os.remove` borra un archivo; se comprueba existencia para evitar error.

11

Enunciado: Renombrar `fichero4.txt` a `renombrado4.txt`.

```
# ejercicio_11.py
import os
if os.path.exists("fichero4.txt"):
    os.rename("fichero4.txt", "renombrado4.txt")
    print("Renombrado fichero4.txt a renombrado4.txt")
else:
    print("fichero4.txt no existe.")
```

Explicación: `os.rename` cambia nombre o mueve si se indica ruta distinta.

12

Enunciado: Obtener el tamaño en bytes de `hola.txt`.

```
# ejercicio_12.py
import os
if os.path.exists("hola.txt"):
    tam = os.path.getsize("hola.txt")
    print(f"hola.txt tiene {tam} bytes")
else:
    print("hola.txt no existe.")
```

Explicación: `os.path.getsize` devuelve tamaño en bytes.

13

Enunciado: Crear un archivo CSV `datos.csv` con 3 filas de datos.

```
# ejercicio_13.py
import csv
with open("datos.csv", "w", newline="", encoding="utf-8") as f:
    writer = csv.writer(f)
    writer.writerow(["nombre", "edad"])
    writer.writerow(["Ana", 30])
    writer.writerow(["Luis", 25])
print("datos.csv creado.")
```

Explicación: Usa módulo `csv` para escribir filas en formato CSV.

14

Enunciado: Leer `datos.csv` y mostrar las filas.

```
# ejercicio_14.py
import csv
with open("datos.csv", "r", encoding="utf-8") as f:
    reader = csv.reader(f)
    for fila in reader:
        print(fila)
```

Explicación: `csv.reader` itera sobre filas del CSV.

15

Enunciado: Guardar un diccionario en `config.json`.

```
# ejercicio_15.py
import json
config = {"usuario": "maria", "tema": "oscuro", "volumen": 75}
with open("config.json", "w", encoding="utf-8") as f:
    json.dump(config, f, ensure_ascii=False, indent=2)
print("config.json guardado.")
```

Explicación: `json.dump` serializa dict a JSON; `ensure_ascii=False` mantiene acentos.

16

Enunciado: Leer y mostrar `config.json`.

```
# ejercicio_16.py
import json
with open("config.json", "r", encoding="utf-8") as f:
    cfg = json.load(f)
print("Contenido de config.json:")
print(cfg)
```

Explicación: `json.load` carga JSON en estructura Python.

17

Enunciado: Comprimir `hola.txt` en `hola.zip`.

```
# ejercicio_17.py
import zipfile
with zipfile.ZipFile("hola.zip", "w") as z:
    z.write("hola.txt")
print("hola.txt comprimido en hola.zip")
```

Explicación: `zipfile.ZipFile` permite crear archivos ZIP; `write` añade archivos.

18

Enunciado: Listar archivos `.txt` en el directorio actual usando `glob`.

```
# ejercicio_18.py
import glob
txts = glob.glob("*.txt")
print("Archivos .txt:", txts)
```

Explicación: `glob.glob` busca por patrón; devuelve lista de coincidencias.

19

Enunciado: Buscar recursivamente archivos `.py` en la carpeta actual y subcarpetas.

```
# ejercicio_19.py
import glob
py_files = glob.glob("**/*.py", recursive=True)
for p in py_files:
    print(p)
```

Explicación: `recursive=True` permite búsqueda en subdirectorios con `**`.

20

Enunciado: Crear un archivo temporal, escribir algo y mostrar su ruta.

```
# ejercicio_20.py
import tempfile
with tempfile.NamedTemporaryFile(delete=False, mode="w", encoding="utf-8") as
tmp:
    tmp.write("Archivo temporal de ejemplo")
    ruta = tmp.name
print("Archivo temporal creado:", ruta)
```

Explicación: `tempfile.NamedTemporaryFile` crea archivo temporal; `delete=False` lo conserva.

21

Enunciado: Comprobar si `mi_carpeta` es un directorio.

```
# ejercicio_21.py
from pathlib import Path
print("mi_carpeta es directorio?", Path("mi_carpeta").is_dir())
```

Explicación: `is_dir()` devuelve True si es directorio.

22

Enunciado: Mostrar la ruta absoluta de `hola.txt`.

```
# ejercicio_22.py
from pathlib import Path
print("Ruta absoluta:", Path("hola.txt").resolve())
```

Explicación: `Path.resolve()` devuelve ruta absoluta canónica.

23

Enunciado: Crear una estructura de carpetas `a/b/c` de forma recursiva.

```
# ejercicio_23.py
from pathlib import Path
Path("a/b/c").mkdir(parents=True, exist_ok=True)
print("Estructura a/b/c creada.")
```

Explicación: `mkdir(parents=True)` crea todos los directorios intermedios.

24

Enunciado: Eliminar una carpeta vacía `temp_dir`.

```
# ejercicio_24.py
import os
try:
    os.rmdir("temp_dir")
    print("temp_dir eliminada.")
except FileNotFoundError:
    print("temp_dir no existe.")
except OSError:
    print("temp_dir no está vacía o no se puede eliminar.")
```

Explicación: `os.rmdir` elimina directorio vacío; si no está vacío lanza OSError.

25

Enunciado: Copiar una carpeta completa `mi_carpeta` a `mi_carpeta_copia`.

```
# ejercicio_25.py
import shutil
shutil.copytree("mi_carpeta", "mi_carpeta_copia", dirs_exist_ok=True)
print("mi_carpeta copiada a mi_carpeta_copia.")
```

Explicación: `shutil.copytree` copia directorios recursivamente; `dirs_exist_ok=True` evita error si exista.

26

Enunciado: Mostrar la fecha de la última modificación de `hola.txt`.

```
# ejercicio_26.py
import os
import datetime
if os.path.exists("hola.txt"):
    ts = os.path.getmtime("hola.txt")
    print("Última modificación:", datetime.datetime.fromtimestamp(ts))
else:
    print("hola.txt no existe.")
```

Explicación: `getmtime` devuelve timestamp de última modificación.

27

Enunciado: Crear un archivo binario `datos.bin` con bytes del 0 al 9.

```
# ejercicio_27.py
with open("datos.bin", "wb") as f:
    f.write(bytes(range(10)))
print("datos.bin creado con bytes 0-9.")
```

Explicación: Modo `"wb"` escribe bytes; `bytes(range(10))` genera 10 bytes.

28

Enunciado: Leer `datos.bin` y mostrar los bytes.

```
# ejercicio_28.py
with open("datos.bin", "rb") as f:
    b = f.read()
print("Bytes leídos:", list(b))
```

Explicación: `"rb"` para leer en binario; convertir a lista para ver valores.

29

Enunciado: Crear un archivo de log `app.log` y escribir líneas con marca de tiempo.

```
# ejercicio_29.py
from datetime import datetime
with open("app.log", "a", encoding="utf-8") as f:
    f.write(f"{datetime.now().isoformat()} - Inicio de la aplicación\n")
print("Línea de log añadida.")
```

Explicación: Append con timestamp ISO para cada entrada.

30

Enunciado: Contar cuántas líneas tiene `datos.csv`.

```
# ejercicio_30.py
count = 0
with open("datos.csv", "r", encoding="utf-8") as f:
    for _ in f:
        count += 1
print(f"datos.csv tiene {count} líneas")
```

Explicación: Iterar sobre archivo cuenta líneas sin cargar todo en memoria.

31

Enunciado: Leer un archivo línea por línea mostrando número de línea.

```
# ejercicio_31.py
with open("hola.txt", "r", encoding="utf-8") as f:
    for i, linea in enumerate(f, 1):
        print(f"{i}: {linea.rstrip()}")
```

Explicación: `enumerate(...,1)` numera desde 1; `rstrip()` quita salto de línea al mostrar.

32

Enunciado: Filtrar las líneas de `datos.csv` que contienen "Ana".

```
# ejercicio_32.py
with open("datos.csv", "r", encoding="utf-8") as f:
    for linea in f:
        if "Ana" in linea:
            print(linea.strip())
```

Explicación: Comprobación simple de substring en cada línea.

33

Enunciado: Crear un archivo `numeros.txt` con los números del 1 al 10, uno por línea.

```
# ejercicio_33.py
with open("numeros.txt", "w", encoding="utf-8") as f:
    for i in range(1, 11):
        f.write(f"{i}\n")
print("numeros.txt creado.")
```

Explicación: Escritura iterativa de números, cada uno en una línea.

34

Enunciado: Sumar los números en `numeros.txt`.

```
# ejercicio_34.py
total = 0
with open("numeros.txt", "r", encoding="utf-8") as f:
    for linea in f:
        total += int(linea.strip())
print("Suma:", total)
```

Explicación: Convertir cada línea a entero y sumar.

35

Enunciado: Crear un archivo `index.html` con un contenido HTML básico.

```
# ejercicio_35.py
html = "<!doctype html>\n<html><body><h1>Hola</h1></body></html>"
with open("index.html", "w", encoding="utf-8") as f:
    f.write(html)
print("index.html creado.")
```

Explicación: Escritura simple de contenido HTML en archivo.

36

Enunciado: Mostrar la extensión de los archivos en el directorio actual.

```
# ejercicio_36.py
import os
exts = set()
for nombre in os.listdir("."):
    if os.path.isfile(nombre):
        _, ext = os.path.splitext(nombre)
        exts.add(ext)
print("Extensiones encontradas:", exts)
```

Explicación: `os.path.splitext` separa nombre y extensión; se usan `set` para unicidad.

37

Enunciado: Leer `datos.csv` usando `DictReader` y mostrar cada fila como dict.

```
# ejercicio_37.py
import csv
with open("datos.csv", "r", encoding="utf-8") as f:
    reader = csv.DictReader(f)
    for row in reader:
        print(row)
```

Explicación: `DictReader` devuelve cada fila como diccionario con cabeceras.

38

Enunciado: Guardar una lista de diccionarios en `personas.json`.

```
# ejercicio_38.py
import json
personas = [{"nombre":"Ana","edad":30}, {"nombre":"Luis","edad":25}]
with open("personas.json", "w", encoding="utf-8") as f:
    json.dump(personas, f, ensure_ascii=False, indent=2)
print("personas.json guardado.")
```

Explicación: Serialización de lista en JSON.

39

Enunciado: Verificar permisos de lectura/escritura de `hola.txt`.

```
# ejercicio_39.py
import os
print("Lectura:", os.access("hola.txt", os.R_OK))
print("Escritura:", os.access("hola.txt", os.W_OK))
```

Explicación: `os.access` comprueba permisos del archivo para el usuario actual.

40

Enunciado: Crear un script que haga backup de `config.json` como `config.json.bak`.

```
# ejercicio_40.py
import shutil
shutil.copyfile("config.json", "config.json.bak")
print("Backup creado config.json.bak")
```

Explicación: `shutil.copyfile` copia contenido a nuevo archivo.

41

Enunciado: Leer sólo las primeras 3 líneas de `hola.txt`.

```
# ejercicio_41.py
with open("hola.txt", "r", encoding="utf-8") as f:
    for _ in range(3):
        linea = f.readline()
        if not linea:
            break
        print(linea.rstrip())
```

Explicación: `readline()` lee una línea por llamada; bucle con límite.

42

Enunciado: Reemplazar todas las apariciones de "Hola" por "Hola de nuevo" en `hola.txt` (sobrescribir).

```
# ejercicio_42.py
from pathlib import Path
p = Path("hola.txt")
text = p.read_text(encoding="utf-8")
text = text.replace("Hola", "Hola de nuevo")
p.write_text(text, encoding="utf-8")
print("Reemplazo completado.")
```

Explicación: Leer todo, usar `replace`, y escribir de nuevo.

43

Enunciado: Dividir `numeros.txt` en dos archivos: pares e impares.

```
# ejercicio_43.py
with open("numeros.txt", "r", encoding="utf-8") as f:
    pares = []
    impares = []
    for linea in f:
        n = int(linea.strip())
        (pares if n%2==0 else impares).append(str(n))
with open("pares.txt","w",encoding="utf-8") as p:
    p.write("\n".join(pares))
with open("impares.txt","w",encoding="utf-8") as q:
    q.write("\n".join(impares))
print("Archivos pares.txt e impares.txt creados.")
```

Explicación: Clasificación básica y escritura en archivos separados.

44

Enunciado: Contar cuántos archivos hay en `mi_carpeta`.

```
# ejercicio_44.py
import os
count = sum(1 for _ in os.listdir("mi_carpeta") if
os.path.isfile(os.path.join("mi_carpeta", _)))
print("Número de archivos en mi_carpeta:", count)
```

Explicación: Itera entradas y cuenta sólo archivos (no subdirectorios).

45

Enunciado: Crear un archivo `.gitignore` con algunas reglas básicas.

```
# ejercicio_45.py
contenido = "__pycache__/\n*.pyc\n.env\n*.log\n"
with open(".gitignore", "w", encoding="utf-8") as f:
    f.write(contenido)
print(".gitignore creado.")
```

Explicación: Escribir patrones comunes de exclusión para Git.

46

Enunciado: Mostrar el tipo (archivo/carpeta) de cada elemento en el directorio actual.

```
# ejercicio_46.py
import os
for nombre in os.listdir("."):
    ruta = os.path.join(".", nombre)
    tipo = "directorio" if os.path.isdir(ruta) else "archivo"
    print(nombre, "-", tipo)
```

Explicación: `os.path.isdir` determina si es directorio.

47

Enunciado: Leer un archivo grande en trozos (chunks) de 1024 bytes.

```
# ejercicio_47.py
with open("datos.bin", "rb") as f:
    while True:
        chunk = f.read(1024)
        if not chunk:
            break
        print("Leído chunk de tamaño", len(chunk))
```

Explicación: Leer por bloques evita cargar todo en memoria.

48

Enunciado: Crear un archivo `usuarios.txt` a partir de una lista y evitar duplicados al añadir.

```
# ejercicio_48.py
nuevos = ["ana","luis","ana"]
ruta = "usuarios.txt"
existentes = set()
try:
    with open(ruta, "r", encoding="utf-8") as f:
        existentes = set(line.strip() for line in f)
except FileNotFoundError:
    pass
with open(ruta, "a", encoding="utf-8") as f:
    for u in nuevos:
        if u not in existentes:
            f.write(u+"\n")
            existentes.add(u)
print("Usuarios añadidos evitando duplicados.")
```

Explicación: Lee existentes en conjunto y sólo añade si no está ya presente.

49

Enunciado: Leer todos los archivos `.txt` y contar el total de palabras.

```
# ejercicio_49.py
import glob
total_palabras = 0
for archivo in glob.glob("*.txt"):
    with open(archivo, "r", encoding="utf-8") as f:
        for linea in f:
            total_palabras += len(linea.split())
print("Total de palabras en .txt:", total_palabras)
```

Explicación: Suma palabras por `split()` en cada línea.

50

Enunciado: Crear un archivo de marcador `bookmark.txt` que almacene la última línea leída.

```
# ejercicio_50.py
ruta = "bookmark.txt"
ultima_linea = None
with open("hola.txt", "r", encoding="utf-8") as f:
    for linea in f:
        ultima_linea = linea
if ultima_linea:
    with open(ruta, "w", encoding="utf-8") as b:
        b.write(ultima_linea)
print("Bookmark guardado.")
```

Explicación: Guarda la última línea leída en otro archivo para seguimiento.

51

Enunciado: Crear un pequeño CSV `inventario.csv` y sumar una columna `cantidad`.

```
# ejercicio_51.py
import csv
items = [["producto", "cantidad"], ["Lapiz", "10"], ["Goma", "5"]]
with open("inventario.csv","w",newline="",encoding="utf-8") as f:
    writer = csv.writer(f)
    writer.writerows(items)
total = 0
with open("inventario.csv","r",encoding="utf-8") as f:
    reader = csv.DictReader(f)
    for row in reader:
        total += int(row["cantidad"])
print("Total cantidad:", total)
```

Explicación: Escribe CSV y luego suma la columna `cantidad` usando `DictReader`.

52

Enunciado: Crear una copia comprimida `.zip` de toda la carpeta `mi_carpeta`.

```
# ejercicio_52.py
import shutil
shutil.make_archive("mi_carpeta_backup", "zip", "mi_carpeta")
print("mi_carpeta comprimida a mi_carpeta_backup.zip")
```

Explicación: `shutil.make_archive` crea archivo comprimido de un directorio.

53

Enunciado: Extraer `hola.zip` en la carpeta `extraido`.

```
# ejercicio_53.py
import zipfile
with zipfile.ZipFile("hola.zip", "r") as z:
    z.extractall("extraido")
print("hola.zip extraído en 'extraido'.")
```

Explicación: `extractall` descomprime todo el contenido en la ruta dada.

54

Enunciado: Crear un archivo `registro.txt` y rotarlo si su tamaño supera 1 KB (renombrarlo añadiendo `.old`).

```
# ejercicio_54.py
import os
ruta = "registro.txt"
with open(ruta, "a", encoding="utf-8") as f:
    f.write("Entrada de log\n")
if os.path.exists(ruta) and os.path.getsize(ruta) > 1024:
    os.rename(ruta, ruta + ".old")
    print("registro.txt renombrado a registro.txt.old")
else:
    print("registro.txt actualizado.")
```

Explicación: Ejemplo simple de rotación por tamaño renombrando el archivo.

55

Enunciado: Comparar si dos archivos `a.txt` y `b.txt` tienen el mismo contenido.

```
# ejercicio_55.py
import filecmp
iguales = filecmp.cmp("a.txt", "b.txt", shallow=False)
print("a.txt y b.txt iguales?", iguales)
```

Explicación: `filecmp.cmp` compara contenido; `shallow=False` hace comparación profunda.

56

Enunciado: Escribir y leer usando `pathlib` en vez de `open`.

```
# ejercicio_56.py
from pathlib import Path
p = Path("ej56.txt")
p.write_text("Texto con pathlib\n", encoding="utf-8")
print("Escrito:", p.read_text(encoding="utf-8"))
```

Explicación: `Path.write_text` y `read_text` son atajos convenientes.

57

Enunciado: Mostrar rutas relativas y absolutas de todos los archivos en `mi_carpeta`.

```
# ejercicio_57.py
from pathlib import Path
p = Path("mi_carpeta")
for f in p.iterdir():
    print("Rel:", f, "Abs:", f.resolve())
```

Explicación: `iterdir()` lista entradas del directorio; `resolve()` da ruta absoluta.

58

Enunciado: Crear un archivo de texto y leerlo usando `with` y `readlines()`.

```
# ejercicio_58.py
with open("ej58.txt","w",encoding="utf-8") as f:
    f.writelines(["Linea1\n","Linea2\n"])
with open("ej58.txt","r",encoding="utf-8") as f:
    lineas = f.readlines()
print(lineas)
```

Explicación: `writelines` escribe lista de líneas; `readlines` devuelve lista de líneas.

59

Enunciado: Buscar archivos que empiecen por `data_` en el directorio actual.

```
# ejercicio_59.py
import glob
for f in glob.glob("data_*"):
    print(f)
```

Explicación: Patrón `data_*` encuentra archivos que comienzan por `data_`.

60

Enunciado: Eliminar todos los archivos `.tmp` en el directorio actual.

```
# ejercicio_60.py
import glob, os
for f in glob.glob("*.tmp"):
    os.remove(f)
print("Archivos .tmp eliminados.")
```

Explicación: Búsqueda por patrón y eliminación iterativa.

61

Enunciado: Crear un pequeño índice `index.txt` con todos los nombres de archivos `.py`.

```
# ejercicio_61.py
import glob
with open("index.txt","w",encoding="utf-8") as idx:
    for f in glob.glob("*.py"):
        idx.write(f + "\n")
print("index.txt creado con .py")
```

Explicación: Útil para crear listados de archivos por tipo.

62

Enunciado: Leer `datos.csv` y convertirlo a JSON `datos.json`.

```
# ejercicio_62.py
import csv, json
with open("datos.csv", "r", encoding="utf-8") as f:
    reader = csv.DictReader(f)
    rows = list(reader)
with open("datos.json", "w", encoding="utf-8") as g:
    json.dump(rows, g, ensure_ascii=False, indent=2)
print("datos.csv convertido a datos.json")
```

Explicación: Transformación CSV → JSON usando diccionarios por fila.

63

Enunciado: Crear un archivo `contador.txt` que incremente un número cada vez que se ejecute el script.

```
# ejercicio_63.py
from pathlib import Path
p = Path("contador.txt")
if p.exists():
    n = int(p.read_text(encoding="utf-8").strip() or "0") + 1
else:
    n = 1
p.write_text(str(n), encoding="utf-8")
print("Contador:", n)
```

Explicación: Lee valor, incrementa y lo escribe de nuevo.

64

Enunciado: Guardar una lista en formato `pickle` y leerla.

```
# ejercicio_64.py
import pickle
data = [1,2,3]
with open("data.pkl","wb") as f:
    pickle.dump(data,f)
with open("data.pkl","rb") as f:
    loaded = pickle.load(f)
print(loaded)
```

Explicación: `pickle` serializa objetos Python en binario. (Uso simple, no para datos no confiables).

65

Enunciado: Buscar y mostrar archivos mayores a 1 KB en el directorio actual.

```
# ejercicio_65.py
import os
for f in os.listdir("."):
    if os.path.isfile(f) and os.path.getsize(f) > 1024:
        print(f, os.path.getsize(f))
```

Explicación: Filtrado por tamaño con `getsize`.

66

Enunciado: Crear un archivo `mayusculas.txt` que contenga el contenido de `hola.txt` en mayúsculas.

```
# ejercicio_66.py
from pathlib import Path
text = Path("hola.txt").read_text(encoding="utf-8")
Path("mayusculas.txt").write_text(text.upper(), encoding="utf-8")
print("mayusculas.txt creado.")
```

Explicación: Transformación de texto y escritura en nuevo archivo.

67

Enunciado: Contar cuántos archivos ocultos (empiezan con `.`) hay en el directorio actual.

```
# ejercicio_67.py
import os
count = sum(1 for f in os.listdir(".") if f.startswith("."))
print("Archivos/carpetas ocultos:", count)
```

Explicación: En Unix-like los archivos ocultos comienzan con `.`.

68

Enunciado: Crear un archivo `tabla.txt` con una tabla 5x5 separada por comas.

```
# ejercicio_68.py
with open("tabla.txt","w",encoding="utf-8") as f:
    for i in range(5):
        f.write(",".join(str(i*j) for j in range(5)) + "\n")
print("tabla.txt creado.")
```

Explicación: Generación de tabla con `join` para separar por comas.

69

Enunciado: Leer `tabla.txt` y mostrar la suma de cada fila.

```
# ejercicio_69.py
with open("tabla.txt","r",encoding="utf-8") as f:
    for linea in f:
        nums = [int(x) for x in linea.split(",") if x.strip()]
        print("Suma fila:", sum(nums))
```

Explicación: Parseo de cada línea como lista de enteros y suma.

70

Enunciado: Crear un script que liste archivos y su tamaño en KB formateado.

```
# ejercicio_70.py
import os
for f in os.listdir("."):
    if os.path.isfile(f):
        kb = os.path.getsize(f)/1024
        print(f"{f}: {kb:.2f} KB")
```

Explicación: Conversión de bytes a kilobytes y formateo con 2 decimales.

71

Enunciado: Comprobar si dos directorios tienen el mismo número de archivos.

```
# ejercicio_71.py
import os
d1 = "mi_carpeta"
d2 = "mi_carpeta_copia"
n1 = sum(1 for _ in os.listdir(d1) if os.path.isfile(os.path.join(d1,_)))
n2 = sum(1 for _ in os.listdir(d2) if os.path.isfile(os.path.join(d2,_)))
print(n1, "vs", n2)
```

Explicación: Cuenta archivos en dos carpetas y muestra comparación.

72

Enunciado: Crear un archivo `fechas.txt` con la fecha actual cada vez que se ejecute.

```
# ejercicio_72.py
from datetime import datetime
with open("fechas.txt","a",encoding="utf-8") as f:
    f.write(datetime.now().isoformat() + "\n")
print("Fecha añadida a fechas.txt")
```

Explicación: Añade timestamp ISO cada ejecución.

73

Enunciado: Ordenar los archivos `.txt` por tamaño y mostrarlos.

```
# ejercicio_73.py
import glob, os
txts = glob.glob("*.txt")
txts.sort(key=lambda f: os.path.getsize(f))
for f in txts:
    print(f, os.path.getsize(f))
```

Explicación: `sort` con key que usa `getsize`.

74

Enunciado: Generar 10 archivos con contenido aleatorio (números) y después borrarlos.

```
# ejercicio_74.py
import random, os
nombres = []
for i in range(10):
    name = f"rnd_{i}.txt"
    with open(name,"w",encoding="utf-8") as f:
        f.write(str(random.randint(0,100)))
    nombres.append(name)
print("Archivos creados:", nombres)
for name in nombres:
    os.remove(name)
print("Archivos borrados.")
```

Explicación: Crea y elimina archivos demostrando flujo completo.

75

Enunciado: Guardar y leer un diccionario usando `shelve`.

```
# ejercicio_75.py
import shelve
with shelve.open("mi_shelve") as db:
    db["clave"] = {"a":1}
with shelve.open("mi_shelve") as db:
    print(db["clave"])
```

Explicación: `shelve` permite almacenar objetos persistentes similar a dict en disco.

76

Enunciado: Mostrar los 5 archivos más recientes por fecha de modificación.

```
# ejercicio_76.py
import os
files = [f for f in os.listdir(".") if os.path.isfile(f)]
files.sort(key=lambda f: os.path.getmtime(f), reverse=True)
for f in files[:5]:
    print(f, os.path.getmtime(f))
```

Explicación: Ordena por `getmtime` y muestra los primeros 5.

77

Enunciado: Reemplazar múltiples archivos `.txt` agregando un prefijo `old_` a su nombre.

```
# ejercicio_77.py
import glob, os
for f in glob.glob("*.txt"):
    os.rename(f, "old_" + f)
print("Archivos .txt renombrados con prefijo old_.")
```

Explicación: Renombrado masivo usando `glob` y `os.rename`.

78

Enunciado: Leer y mostrar sólo las líneas que no estén vacías de `ej58.txt`.

```
# ejercicio_78.py
with open("ej58.txt","r",encoding="utf-8") as f:
    for linea in f:
        if linea.strip():
            print(linea.strip())
```

Explicación: `strip()` elimina espacios; se filtran líneas vacías.

79

Enunciado: Crear un script que genere un nombre de archivo único usando fecha y hora.

```
# ejercicio_79.py
from datetime import datetime
nombre = "backup_" + datetime.now().strftime("%Y%m%d_%H%M%S") + ".txt"
with open(nombre,"w",encoding="utf-8") as f:
    f.write("Backup creado")
print("Archivo creado:", nombre)
```

Explicación: `strftime` para generar timestamp usable en nombres.

80

Enunciado: Copiar y renombrar todos los `.py` añadiendo sufijo `_copy`.

```
# ejercicio_80.py
import glob, shutil, os
for f in glob.glob("*.py"):
    destino = os.path.splitext(f)[0] + "_copy.py"
    shutil.copy(f, destino)
print("Copias de .py creadas con sufijo _copy.")
```

Explicación: Copia y crea nuevo nombre manteniendo extensión.

81

Enunciado: Escribir y leer un archivo UTF-8 con caracteres especiales (acentos, ñ).

```
# ejercicio_81.py
texto = "Español: ñ, á, é, í, ó, ú"
with open("utf8.txt","w",encoding="utf-8") as f:
    f.write(texto)
with open("utf8.txt","r",encoding="utf-8") as f:
    print(f.read())
```

Explicación: Demuestra `encoding="utf-8"` para caracteres no ASCII.

82

Enunciado: Mostrar la extensión mayoritaria de archivos en el directorio.

```
# ejercicio_82.py
import os
from collections import Counter
exts = [os.path.splitext(f)[1] for f in os.listdir(".") if os.path.isfile(f)]
most = Counter(exts).most_common(1)
print("Extensión mayoritaria:", most[0] if most else None)
```

Explicación: `Counter` para contar y obtener la más frecuente.

83

Enunciado: Guardar una lista de strings en un archivo usando `writelines` asegurando saltos de línea.

```
# ejercicio_83.py
lineas = ["primera\n","segunda\n","tercera\n"]
with open("lista.txt","w",encoding="utf-8") as f:
    f.writelines(lineas)
print("lista.txt creado con writelines.")
```

Explicación: `writelines` no añade saltos por sí mismo; deben incluirse.

84

Enunciado: Crear un script que lea un archivo y haga copia de seguridad con sufijo `.bak` antes de modificarlo.

```
# ejercicio_84.py
from pathlib import Path
p = Path("hola.txt")
if p.exists():
    p.rename(p.with_suffix(p.suffix + ".bak"))
    p.write_text("Contenido nuevo", encoding="utf-8")
    print("Backup creado y archivo modificado.")
else:
    print("hola.txt no existe.")
```

Explicación: Renombra original a `.bak` y escribe nuevo contenido.

85

Enunciado: Guardar líneas de un archivo en una lista y eliminar duplicados manteniendo orden.

```
# ejercicio_85.py
seen = set()
unique = []
with open("hola.txt","r",encoding="utf-8") as f:
    for line in f:
        s = line.rstrip("\n")
        if s not in seen:
            seen.add(s)
            unique.append(s)
print(unique)
```

Explicación: Usa `set` para seguimiento y lista para mantener orden.

86

Enunciado: Crear un script que devuelva el nombre del archivo más grande.

```
# ejercicio_86.py
import os
files = [f for f in os.listdir(".") if os.path.isfile(f)]
if files:
    mayor = max(files, key=lambda f: os.path.getsize(f))
    print("Archivo más grande:", mayor, os.path.getsize(mayor))
else:
    print("No hay archivos.")
```

Explicación: `max` con `key` para tamaño devuelve archivo mayor.

87

Enunciado: Convertir todos los nombres de archivo a minúsculas en el directorio actual.

```
# ejercicio_87.py
import os
for f in os.listdir("."):
    if os.path.isfile(f):
        new = f.lower()
        if new != f:
            os.rename(f, new)
print("Nombres convertidos a minúsculas.")
```

Explicación: Renombrado condicional evitando conflicto con mismo nombre.

88

Enunciado: Hacer un listado recursivo de archivos y guardarlo en `arbol.txt`.

```
# ejercicio_88.py
import os
with open("arbol.txt","w",encoding="utf-8") as out:
    for root, dirs, files in os.walk("."):
        out.write(root + "\n")
        for f in files:
            out.write("  " + f + "\n")
print("Listado recursivo guardado en arbol.txt")
```

Explicación: `os.walk` recorre directorios recursivamente.

89

Enunciado: Detectar si un archivo está vacío (0 bytes).

```
# ejercicio_89.py
import os
ruta = "ej58.txt"
if os.path.exists(ruta) and os.path.getsize(ruta) == 0:
    print(ruta, "está vacío")
else:
    print(ruta, "no está vacío o no existe")
```

Explicación: `getsize` == 0 indica archivo vacío.

90

Enunciado: Reescribir un archivo eliminando líneas que contengan la palabra "ERROR".

```
# ejercicio_90.py
from pathlib import Path
p = Path("registro.txt")
if p.exists():
    lines = p.read_text(encoding="utf-8").splitlines()
    filtered = [l for l in lines if "ERROR" not in l]
    p.write_text("\n".join(filtered) + ("\n" if filtered else ""),
encoding="utf-8")
    print("Registro limpiado de líneas con ERROR.")
else:
    print("registro.txt no existe.")
```

Explicación: Filtrado en memoria y sobrescritura del archivo.

91

Enunciado: Crear un script que lea un archivo y muestre sólo las 10 últimas líneas (like `tail`).

```
# ejercicio_91.py
from collections import deque
with open("registro.txt","r",encoding="utf-8") as f:
    ultimas = deque(f, maxlen=10)
for l in ultimas:
    print(l.rstrip())
```

Explicación: `deque` con `maxlen` mantiene sólo últimas N líneas eficientemente.

92

Enunciado: Crear `README.md` con un texto descriptivo del proyecto.

```
# ejercicio_92.py
texto = "# Proyecto de ejemplo\nEste proyecto contiene ejercicios sobre
ficheros.\n"
with open("README.md","w",encoding="utf-8") as f:
    f.write(texto)
print("README.md creado.")
```

Explicación: Escritura de markdown para documentación básica.

93

Enunciado: Cambiar la codificación de un archivo de `latin-1` a `utf-8`.

```
# ejercicio_93.py
from pathlib import Path
p = Path("latin1.txt")
if p.exists():
    text = p.read_text(encoding="latin-1")
    p.write_text(text, encoding="utf-8")
    print("Archivo convertido a utf-8.")
else:
    print("latin1.txt no existe.")
```

Explicación: Leer con la codificación original y reescribir en UTF-8.

94

Enunciado: Crear un archivo `hashes.txt` con el hash MD5 de cada archivo `.txt`.

```
# ejercicio_94.py
import hashlib, glob
with open("hashes.txt","w",encoding="utf-8") as out:
    for f in glob.glob("*.txt"):
        h = hashlib.md5(open(f,"rb").read()).hexdigest()
        out.write(f"{f} {h}\n")
print("hashes.txt creado.")
```

Explicación: Calcula MD5 leyendo archivos en binario; útil para detectar cambios.

95

Enunciado: Crear un script que detecte archivos duplicados por tamaño y hash.

```
# ejercicio_95.py
import os, hashlib
by_size = {}
for f in os.listdir("."):
    if os.path.isfile(f):
        sz = os.path.getsize(f)
        by_size.setdefault(sz, []).append(f)
duplicates = []
for size, files in by_size.items():
    if len(files) > 1:
        hashes = {}
        for f in files:
            h = hashlib.md5(open(f,"rb").read()).hexdigest()
            hashes.setdefault(h, []).append(f)
        for group in hashes.values():
            if len(group) > 1:
                duplicates.append(group)
print("Duplicados detectados:", duplicates)
```

Explicación: Agrupa por tamaño, luego por hash para confirmar duplicados.

96

Enunciado: Guardar un log de operaciones (crear, borrar) en `operaciones.log` cada vez que se realice una acción.

```
# ejercicio_96.py
from datetime import datetime
def log(op, ruta):
    with open("operaciones.log","a",encoding="utf-8") as f:
        f.write(f"{datetime.now().isoformat()} - {op} - {ruta}\n")
# Ejemplo de uso:
open("temp_op.txt","w").close()
log("CREAR", "temp_op.txt")
import os
os.remove("temp_op.txt")
log("BORRAR", "temp_op.txt")
print("Operaciones registradas.")
```

Explicación: Función `log` centraliza escritura en archivo de operaciones.

97

Enunciado: Comprimir varios archivos seleccionados en un zip `seleccion.zip`.

```
# ejercicio_97.py
import zipfile
files = ["hola.txt","datos.csv","config.json"]
with zipfile.ZipFile("seleccion.zip","w") as z:
    for f in files:
        try:
            z.write(f)
        except FileNotFoundError:
            pass
print("seleccion.zip creado con los archivos existentes.")
```

Explicación: Añade sólo archivos existentes, evitando errores con `try/except`.

98

Enunciado: Crear un script que convierta todas las comas en `,` por punto y coma `;` en archivos `.csv`.

```
# ejercicio_98.py
import glob
for f in glob.glob("*.csv"):
    with open(f,"r",encoding="utf-8") as fh:
        data = fh.read()
    data = data.replace(",", ";")
    with open(f,"w",encoding="utf-8") as fh:
        fh.write(data)
print("Reemplazo de comas por puntos y comas en CSV completado.")
```

Explicación: Reemplazo simple de caracteres en archivos CSV (cuidado con comillas en CSV real).

99

Enunciado: Mostrar la cantidad total de bytes usados por los archivos en el directorio actual.

```
# ejercicio_99.py
import os
total = sum(os.path.getsize(f) for f in os.listdir(".") if os.path.isfile(f))
print("Total de bytes en el directorio:", total)
```

Explicación: Suma de tamaños de todos los archivos para conocer uso de espacio.

100

Enunciado: Generar un resumen `resumen.txt` que contenga: número de archivos, número de directorios y tamaño total.

```
# ejercicio_100.py
import os
num_files = 0
num_dirs = 0
total_size = 0
for root, dirs, files in os.walk("."):
    num_dirs += len(dirs)
    for f in files:
        num_files += 1
        total_size += os.path.getsize(os.path.join(root, f))
with open("resumen.txt","w",encoding="utf-8") as r:
    r.write(f"Archivos: {num_files}\nDirectorios: {num_dirs}\nTamaño total:
{total_size} bytes\n")
print("resumen.txt creado.")
```

Explicación: `os.walk` recorre todo; contabiliza archivos, directorios y suma tamaños.

Ejercicios python archivos csv

Ejercicio 1

Enunciado: Crear un archivo CSV vacío.
Explicación: Usamos la función `open()` con el modo `"w"` para crear un archivo nuevo.

```
import csv

with open("archivo1.csv", "w", newline="") as archivo:
    pass  # Crea un archivo vacío
```

Ejercicio 2

Enunciado: Escribir una sola fila en un archivo CSV.
Explicación: Usamos `csv.writer()` para escribir filas en un CSV.

```
import csv

with open("archivo2.csv", "w", newline="") as archivo:
    escritor = csv.writer(archivo)
    escritor.writerow(["Nombre", "Edad", "Ciudad"])
```

Ejercicio 3

Enunciado: Escribir varias filas en un archivo CSV.
Explicación: `writerows()` permite escribir varias filas de datos a la vez.

```
import csv

with open("archivo3.csv", "w", newline="") as archivo:
    escritor = csv.writer(archivo)
    filas = [
        ["Nombre", "Edad", "Ciudad"],
        ["Ana", 25, "Zaragoza"],
        ["Luis", 30, "Madrid"]
    ]
    escritor.writerows(filas)
```

Ejercicio 4

Enunciado: Leer un archivo CSV y mostrar su contenido.
Explicación: `csv.reader()` permite leer cada fila del CSV como lista.

```
import csv

with open("archivo3.csv", "r") as archivo:
    lector = csv.reader(archivo)
    for fila in lector:
        print(fila)
```

Ejercicio 5

Enunciado: Leer un CSV y mostrar solo la primera columna.
Explicación: Iteramos sobre las filas y mostramos el primer elemento de cada lista.

```
import csv

with open("archivo3.csv", "r") as archivo:
    lector = csv.reader(archivo)
    for fila in lector:
        print(fila[0])
```

Ejercicio 6

Enunciado: Agregar una fila al final de un CSV existente.
Explicación: Usamos el modo `"a"` para agregar datos sin sobrescribir el archivo.

```
import csv

with open("archivo3.csv", "a", newline="") as archivo:
    escritor = csv.writer(archivo)
    escritor.writerow(["Marta", 28, "Barcelona"])
```

Ejercicio 7

Enunciado: Leer un CSV y convertirlo en una lista de diccionarios.
Explicación: `csv.DictReader()` permite acceder a los datos usando los nombres de columna.

```
import csv

with open("archivo3.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        print(fila)
```

Ejercicio 8

Enunciado: Escribir un CSV usando diccionarios.
Explicación: `csv.DictWriter()` permite escribir datos usando claves que coinciden con los nombres de columna.

```
import csv

with open("archivo4.csv", "w", newline="") as archivo:
    campos = ["Nombre", "Edad", "Ciudad"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()
    escritor.writerow({"Nombre": "Ana", "Edad": 25, "Ciudad":
"Zaragoza"})
```

Ejercicio 9

Enunciado: Escribir varias filas usando diccionarios.
Explicación: `writerows()` permite escribir múltiples diccionarios a la vez.

```
import csv

with open("archivo4.csv", "w", newline="") as archivo:
    campos = ["Nombre", "Edad", "Ciudad"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()
    filas = [
        {"Nombre": "Ana", "Edad": 25, "Ciudad": "Zaragoza"},
        {"Nombre": "Luis", "Edad": 30, "Ciudad": "Madrid"}
    ]
    escritor.writerows(filas)
```

Ejercicio 10

Enunciado: Leer un CSV como diccionario y mostrar solo una columna.
Explicación: Iteramos sobre cada fila y accedemos a los datos por clave.

```
import csv

with open("archivo4.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        print(fila["Nombre"])
```

Ejercicio 11

Enunciado: Leer un CSV y contar cuántas filas tiene.
Explicación: Iteramos sobre el lector y usamos un contador para obtener el número de filas.

```
import csv

contador = 0
with open("archivo3.csv", "r") as archivo:
    lector = csv.reader(archivo)
    for fila in lector:
        contador += 1
print(f"El archivo tiene {contador} filas")
```

Ejercicio 12

Enunciado: Leer un CSV y contar cuántas columnas tiene.
Explicación: Leemos la primera fila y usamos `len()` para obtener el número de columnas.

```
import csv

with open("archivo3.csv", "r") as archivo:
    lector = csv.reader(archivo)
    primera_fila = next(lector)
    print(f"El archivo tiene {len(primera_fila)} columnas")
```

Ejercicio 13

Enunciado: Leer un CSV y mostrar solo filas donde la edad sea mayor a 25.
Explicación: Convertimos la columna de edad a entero y filtramos según la condición.

```
import csv

with open("archivo3.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if int(fila["Edad"]) > 25:
            print(fila)
```

Ejercicio 14

Enunciado: Crear un CSV y escribir datos numéricos con formato.
Explicación: Podemos escribir números y darles formato al guardarlos.

```
import csv

with open("archivo5.csv", "w", newline="") as archivo:
    escritor = csv.writer(archivo)
    escritor.writerow(["Producto", "Precio"])
    escritor.writerow(["Manzana", "{:.2f}".format(0.857)])
    escritor.writerow(["Banana", "{:.2f}".format(1.234)])
```

Ejercicio 15

Enunciado: Leer un CSV y sumar los valores de una columna numérica.
Explicación: Iteramos y sumamos los valores de la columna convertidos a float.

```
import csv

suma = 0
with open("archivo5.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        suma += float(fila["Precio"])
print(f"Suma de precios: {suma}")
```

Ejercicio 16

Enunciado: Leer un CSV y mostrar el promedio de una columna.
Explicación: Sumamos los valores y los dividimos por el número de filas.

```
import csv

total = 0
contador = 0
with open("archivo5.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        total += float(fila["Precio"])
        contador += 1
promedio = total / contador
print(f"Promedio de precios: {promedio}")
```

Ejercicio 17

Enunciado: Leer un CSV y mostrar los valores máximos de una columna.
Explicación: Usamos una variable para ir comparando y guardando el máximo.

```
import csv

maximo = 0
with open("archivo5.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        precio = float(fila["Precio"])
        if precio > maximo:
            maximo = precio
print(f"Precio máximo: {maximo}")
```

Ejercicio 18

Enunciado: Leer un CSV y mostrar los valores mínimos de una columna.
Explicación: Inicializamos con `None` y vamos comparando para obtener el mínimo.

```
import csv

minimo = None
with open("archivo5.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        precio = float(fila["Precio"])
        if minimo is None or precio < minimo:
            minimo = precio
print(f"Precio mínimo: {minimo}")
```

Ejercicio 19

Enunciado: Filtrar un CSV y crear uno nuevo solo con filas que cumplen una condición.
Explicación: Leemos un CSV, filtramos filas y las escribimos en un nuevo archivo.

```
import csv

with open("archivo3.csv", "r") as entrada,
open("archivo_filtrado.csv", "w", newline="") as salida:
    lector = csv.DictReader(entrada)
    campos = lector.fieldnames
    escritor = csv.DictWriter(salida, fieldnames=campos)
    escritor.writeheader()
    for fila in lector:
        if int(fila["Edad"]) > 25:
            escritor.writerow(fila)
```

Ejercicio 20

Enunciado: Contar cuántas veces aparece un valor en una columna de CSV.
Explicación: Usamos un contador y comparamos cada fila con el valor deseado.

```
import csv

contador = 0
with open("archivo3.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if fila["Ciudad"] == "Madrid":
            contador += 1
print(f"Veces que aparece Madrid: {contador}")
```

Ejercicio 21

Enunciado: Leer un CSV y mostrar solo las filas que contienen un nombre específico.
Explicación: Filtramos las filas comprobando si el valor de la columna coincide con el nombre buscado.

```
import csv

nombre_buscado = "Ana"
with open("archivo3.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if fila["Nombre"] == nombre_buscado:
            print(fila)
```

Ejercicio 22

Enunciado: Ordenar los datos de un CSV por una columna numérica.
Explicación: Leemos todas las filas en una lista y usamos `sorted()` con `key` para ordenar.

```
import csv

with open("archivo3.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    filas = list(lector)
    filas_ordenadas = sorted(filas, key=lambda x: int(x["Edad"]))
    for fila in filas_ordenadas:
        print(fila)
```

Ejercicio 23

Enunciado: Agregar varias filas a un CSV existente.
Explicación: Usamos el modo `"a"` para no sobrescribir el archivo y escribimos varias filas.

```
import csv

with open("archivo3.csv", "a", newline="") as archivo:
    escritor = csv.writer(archivo)
    escritor.writerows([
        ["Carlos", 22, "Sevilla"],
        ["Lucía", 27, "Valencia"]
    ])
```

Ejercicio 24

Enunciado: Leer un CSV y mostrar los nombres únicos de una columna.
Explicación: Usamos un conjunto (`set`) para almacenar solo valores únicos.

```
import csv

nombres = set()
with open("archivo3.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        nombres.add(fila["Nombre"])
print(nombres)
```

Ejercicio 25

Enunciado: Contar cuántos valores distintos hay en una columna.
Explicación: Convertimos los valores a un conjunto y contamos su tamaño.

```
import csv

ciudades = set()
with open("archivo3.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        ciudades.add(fila["Ciudad"])
print(f"Número de ciudades distintas: {len(ciudades)}")
```

Ejercicio 26

Enunciado: Leer un CSV y mostrar los valores de una columna específica como lista.
Explicación: Iteramos sobre el CSV y guardamos los valores deseados en una lista.

```
import csv

edades = []
with open("archivo3.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        edades.append(int(fila["Edad"]))
print(edades)
```

Ejercicio 27

Enunciado: Crear un CSV con encabezados y varias filas usando lista de diccionarios.
Explicación: `DictWriter` permite escribir directamente diccionarios.

```
import csv

datos = [
    {"Nombre": "Ana", "Edad": 25, "Ciudad": "Zaragoza"},
    {"Nombre": "Luis", "Edad": 30, "Ciudad": "Madrid"}
]

with open("archivo6.csv", "w", newline="") as archivo:
    campos = ["Nombre", "Edad", "Ciudad"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()
    escritor.writerows(datos)
```

Ejercicio 28

Enunciado: Leer un CSV y convertir una columna en lista de enteros.
Explicación: Tomamos cada valor de la columna y lo convertimos a entero antes de guardarlo en la lista.

```
import csv

edades = []
with open("archivo6.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        edades.append(int(fila["Edad"]))
print(edades)
```

Ejercicio 29

Enunciado: Filtrar filas de un CSV por ciudad y crear una nueva lista con esos datos.
Explicación: Comprobamos el valor de la columna ciudad y guardamos las filas que coinciden.

```
import csv

filas_filtradas = []
with open("archivo6.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if fila["Ciudad"] == "Madrid":
            filas_filtradas.append(fila)
print(filas_filtradas)
```

Ejercicio 30

Enunciado: Leer un CSV y mostrar la fila con la mayor edad.
Explicación: Comparamos cada fila y guardamos la que tenga la edad máxima.

```
import csv

max_fila = None
with open("archivo6.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if max_fila is None or int(fila["Edad"]) >
int(max_fila["Edad"]):
            max_fila = fila
print(max_fila)
```

Ejercicio 31

Enunciado: Crear un CSV y escribir filas usando un bucle.
Explicación: Se pueden generar filas automáticamente usando un bucle `for`.

```
import csv

with open("archivo7.csv", "w", newline="") as archivo:
    escritor = csv.writer(archivo)
    escritor.writerow(["Número", "Cuadrado"])
    for i in range(1, 6):
        escritor.writerow([i, i**2])
```

Ejercicio 32

Enunciado: Leer un CSV y mostrar solo los valores de la columna "Cuadrado".
Explicación: Iteramos sobre cada fila y accedemos a la columna deseada.

```
import csv

with open("archivo7.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        print(fila["Cuadrado"])
```

Ejercicio 33

Enunciado: Crear un CSV con encabezados y varios datos usando lista de diccionarios.
Explicación: `DictWriter` permite escribir datos directamente con claves y valores.

```
import csv

datos = [
    {"Producto": "Manzana", "Precio": 0.85},
    {"Producto": "Banana", "Precio": 1.25},
    {"Producto": "Cereza", "Precio": 2.50}
]

with open("archivo8.csv", "w", newline="") as archivo:
    campos = ["Producto", "Precio"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()
    escritor.writerows(datos)
```

Ejercicio 34

Enunciado: Leer un CSV y calcular el precio total de todos los productos.
Explicación: Sumamos los valores de la columna "Precio".

```
import csv

total = 0
with open("archivo8.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        total += float(fila["Precio"])
print(f"Precio total: {total}")
```

Ejercicio 35

Enunciado: Crear un CSV y guardar datos con encabezados y un bucle.
Explicación: Se puede generar automáticamente filas usando `range()` y `DictWriter`.

```
import csv

with open("archivo9.csv", "w", newline="") as archivo:
    campos = ["Número", "Doble"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()
    for i in range(1, 6):
        escritor.writerow({"Número": i, "Doble": i*2})
```

Ejercicio 36

Enunciado: Leer un CSV y mostrar la fila con el valor máximo en una columna.
Explicación: Iteramos comparando los valores para encontrar el máximo.

```
import csv

max_fila = None
with open("archivo9.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if max_fila is None or int(fila["Doble"]) >
int(max_fila["Doble"]):
            max_fila = fila
print(max_fila)
```

Ejercicio 37

Enunciado: Leer un CSV y mostrar la fila con el valor mínimo en una columna.
Explicación: Iteramos comparando los valores para encontrar el mínimo.

```
import csv

min_fila = None
with open("archivo9.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if min_fila is None or int(fila["Doble"]) <
int(min_fila["Doble"]):
            min_fila = fila
print(min_fila)
```

Ejercicio 38

Enunciado: Crear un CSV y escribir filas con números pares usando bucle.
Explicación: Solo guardamos los números que cumplen la condición de par.

```
import csv

with open("archivo10.csv", "w", newline="") as archivo:
    escritor = csv.writer(archivo)
    escritor.writerow(["Número", "Par"])
    for i in range(1, 11):
        if i % 2 == 0:
            escritor.writerow([i, True])
```

Ejercicio 39

Enunciado: Leer un CSV y mostrar solo los números pares.
Explicación: Filtramos las filas comprobando la columna "Par".

```
import csv

with open("archivo10.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if fila["Par"] == "True":
            print(fila["Número"])
```

Ejercicio 40

Enunciado: Crear un CSV y escribir filas con nombres y edades aleatorias.
Explicación: Usamos la librería `random` para generar edades y guardarlas en CSV.

```
import csv
import random

nombres = ["Ana", "Luis", "Marta", "Carlos"]
with open("archivo11.csv", "w", newline="") as archivo:
    escritor = csv.writer(archivo)
    escritor.writerow(["Nombre", "Edad"])
    for nombre in nombres:
        escritor.writerow([nombre, random.randint(20, 40)])
```

Ejercicio 41

Enunciado: Leer un CSV y calcular el promedio de las edades.
Explicación: Sumamos los valores de la columna "Edad" y los dividimos por el número de filas.

```
import csv

total = 0
contador = 0
with open("archivo11.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        total += int(fila["Edad"])
        contador += 1
promedio = total / contador
print(f"Promedio de edades: {promedio}")
```

Ejercicio 42

Enunciado: Filtrar un CSV y crear un nuevo CSV con personas mayores de 25 años.
Explicación: Leemos las filas y escribimos solo las que cumplen la condición en un nuevo archivo.

```
import csv

with open("archivo11.csv", "r") as entrada,
open("archivo_mayores25.csv", "w", newline="") as salida:
    lector = csv.DictReader(entrada)
    campos = lector.fieldnames
    escritor = csv.DictWriter(salida, fieldnames=campos)
    escritor.writeheader()
    for fila in lector:
        if int(fila["Edad"]) > 25:
            escritor.writerow(fila)
```

Ejercicio 43

Enunciado: Leer un CSV y mostrar los nombres en orden alfabético.
Explicación: Guardamos los nombres en una lista y usamos `sorted()`.

```
import csv

nombres = []
with open("archivo11.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        nombres.append(fila["Nombre"])
nombres_ordenados = sorted(nombres)
print(nombres_ordenados)
```

Ejercicio 44

Enunciado: Crear un CSV y escribir filas con números aleatorios.
Explicación: Usamos `random.randint()` para generar los valores.

```
import csv
import random

with open("archivo12.csv", "w", newline="") as archivo:
    escritor = csv.writer(archivo)
    escritor.writerow(["Número", "Valor"])
    for i in range(1, 6):
        escritor.writerow([i, random.randint(10, 50)])
```

Ejercicio 45

Enunciado: Leer un CSV y mostrar el valor máximo de la columna "Valor".
Explicación: Iteramos y guardamos el mayor valor encontrado.

```
import csv

max_valor = None
with open("archivo12.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        valor = int(fila["Valor"])
        if max_valor is None or valor > max_valor:
            max_valor = valor
print(f"Valor máximo: {max_valor}")
```

Ejercicio 46

Enunciado: Leer un CSV y mostrar el valor mínimo de la columna "Valor".
Explicación: Iteramos y guardamos el menor valor encontrado.

```
import csv

min_valor = None
with open("archivo12.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        valor = int(fila["Valor"])
        if min_valor is None or valor < min_valor:
            min_valor = valor
print(f"Valor mínimo: {min_valor}")
```

Ejercicio 47

Enunciado: Crear un CSV y guardar datos de frutas y cantidades usando lista de diccionarios.
Explicación: `DictWriter` permite guardar datos con nombres de columnas como claves.

```
import csv

frutas = [
    {"Fruta": "Manzana", "Cantidad": 10},
    {"Fruta": "Banana", "Cantidad": 15},
    {"Fruta": "Cereza", "Cantidad": 8}
]

with open("archivo13.csv", "w", newline="") as archivo:
    campos = ["Fruta", "Cantidad"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()
    escritor.writerows(frutas)
```

Ejercicio 48

Enunciado: Leer un CSV y calcular la suma de la columna "Cantidad".
Explicación: Iteramos sobre las filas y acumulamos los valores.

```
import csv

total = 0
with open("archivo13.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        total += int(fila["Cantidad"])
print(f"Cantidad total de frutas: {total}")
```

Ejercicio 49

Enunciado: Crear un CSV y escribir datos de empleados con nombre y salario.
Explicación: Guardamos los datos usando `DictWriter`.

```
import csv

empleados = [
    {"Nombre": "Ana", "Salario": 2500},
    {"Nombre": "Luis", "Salario": 3000},
    {"Nombre": "Marta", "Salario": 2800}
]

with open("archivo14.csv", "w", newline="") as archivo:
    campos = ["Nombre", "Salario"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()
    escritor.writerows(empleados)
```

Ejercicio 50

Enunciado: Leer un CSV y mostrar el salario promedio.
Explicación: Sumamos los salarios y dividimos entre el número de filas.

```
import csv

total = 0
contador = 0
with open("archivo14.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        total += int(fila["Salario"])
        contador += 1
promedio = total / contador
print(f"Salario promedio: {promedio}")
```

Si quieres, puedo continuar con el **Bloque 6 (ejercicios 51 al 60)** para seguir avanzando hacia los 100 programas de archivos CSV.

¿Deseas que continúe con el **Bloque 6**?

¡Perfecto! 🌟 Aquí tienes el **Bloque 6 de 10 programas (51 al 60)** sobre **archivos CSV**, nivel fácil, con enunciado, explicación y código:

Ejercicio 51

Enunciado: Filtrar un CSV y mostrar solo los empleados con salario mayor a 2800.
Explicación: Iteramos sobre las filas y comprobamos la columna "Salario".

```
import csv

with open("archivo14.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if int(fila["Salario"]) > 2800:
            print(fila)
```

Ejercicio 52

Enunciado: Leer un CSV y crear una lista de nombres de empleados.
Explicación: Iteramos sobre las filas y extraemos la columna "Nombre".

```
import csv

nombres = []
with open("archivo14.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        nombres.append(fila["Nombre"])
print(nombres)
```

Ejercicio 53

Enunciado: Crear un CSV con números aleatorios y su cuadrado.
Explicación: Generamos números con `random.randint()` y calculamos su cuadrado.

```
import csv
import random

with open("archivo15.csv", "w", newline="") as archivo:
    escritor = csv.writer(archivo)
    escritor.writerow(["Número", "Cuadrado"])
    for _ in range(5):
        n = random.randint(1, 10)
        escritor.writerow([n, n**2])
```

Ejercicio 54

Enunciado: Leer un CSV y mostrar solo los números mayores que 5.
Explicación: Filtramos los valores de la columna "Número" según la condición.

```
import csv

with open("archivo15.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if int(fila["Número"]) > 5:
            print(fila)
```

Ejercicio 55

Enunciado: Crear un CSV con nombres y edades usando bucle.
Explicación: Iteramos sobre listas de nombres y generamos edades aleatorias.

```
import csv
import random

nombres = ["Ana", "Luis", "Marta", "Carlos"]
with open("archivo16.csv", "w", newline="") as archivo:
    escritor = csv.writer(archivo)
    escritor.writerow(["Nombre", "Edad"])
    for nombre in nombres:
        escritor.writerow([nombre, random.randint(20, 40)])
```

Ejercicio 56

Enunciado: Leer un CSV y mostrar el nombre del empleado más joven.
Explicación: Iteramos comparando la columna "Edad" para encontrar el mínimo.

```
import csv

min_fila = None
with open("archivo16.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if min_fila is None or int(fila["Edad"]) <
int(min_fila["Edad"]):
            min_fila = fila
print(min_fila["Nombre"])
```

Ejercicio 57

Enunciado: Crear un CSV con productos y cantidades usando lista de diccionarios.
Explicación: `DictWriter` permite escribir datos con claves que corresponden a columnas.

```
import csv

productos = [
    {"Producto": "Manzana", "Cantidad": 10},
    {"Producto": "Banana", "Cantidad": 15},
    {"Producto": "Cereza", "Cantidad": 8}
]

with open("archivo17.csv", "w", newline="") as archivo:
    campos = ["Producto", "Cantidad"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()
    escritor.writerows(productos)
```

Ejercicio 58

Enunciado: Leer un CSV y calcular la cantidad total de productos.
Explicación: Sumamos los valores de la columna "Cantidad".

```
import csv

total = 0
with open("archivo17.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        total += int(fila["Cantidad"])
print(f"Cantidad total: {total}")
```

Ejercicio 59

Enunciado: Filtrar un CSV y crear uno nuevo solo con productos cuya cantidad sea mayor a 10.
Explicación: Escribimos solo las filas que cumplen la condición en un nuevo archivo.

```
import csv

with open("archivo17.csv", "r") as entrada,
open("archivo17_filtrado.csv", "w", newline="") as salida:
    lector = csv.DictReader(entrada)
    campos = lector.fieldnames
    escritor = csv.DictWriter(salida, fieldnames=campos)
    escritor.writeheader()
    for fila in lector:
        if int(fila["Cantidad"]) > 10:
            escritor.writerow(fila)
```

Ejercicio 60

Enunciado: Leer un CSV y mostrar los productos en orden alfabético.
Explicación: Guardamos los nombres en una lista y usamos `sorted()`.

```
import csv

productos = []
with open("archivo17.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        productos.append(fila["Producto"])
productos_ordenados = sorted(productos)
print(productos_ordenados)
```

Ejercicio 61

Enunciado: Leer un CSV y mostrar las filas con productos cuya cantidad sea menor a 10.
Explicación: Filtramos las filas según la condición en la columna "Cantidad".

```
import csv

with open("archivo17.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if int(fila["Cantidad"]) < 10:
            print(fila)
```

Ejercicio 62

Enunciado: Leer un CSV y crear una lista con los productos y sus cantidades.
Explicación: Iteramos sobre las filas y agregamos tuplas a una lista.

```
import csv

lista_productos = []
with open("archivo17.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        lista_productos.append((fila["Producto"],
int(fila["Cantidad"])))
print(lista_productos)
```

Ejercicio 63

Enunciado: Crear un CSV con números y sus cubos usando bucle.
Explicación: Calculamos el cubo de cada número y lo escribimos en el CSV.

```
import csv

with open("archivo18.csv", "w", newline="") as archivo:
    escritor = csv.writer(archivo)
    escritor.writerow(["Número", "Cubo"])
    for i in range(1, 6):
        escritor.writerow([i, i**3])
```

Ejercicio 64

Enunciado: Leer un CSV y mostrar los números cuyo cubo sea mayor que 20.
Explicación: Filtramos las filas según la condición en la columna "Cubo".

```
import csv

with open("archivo18.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if int(fila["Cubo"]) > 20:
            print(fila)
```

Ejercicio 65

Enunciado: Crear un CSV con nombres y edades, y guardar filas usando bucle y lista de diccionarios.
Explicación: Generamos diccionarios dinámicamente y los escribimos con `DictWriter`.

```
import csv

nombres = ["Ana", "Luis", "Marta", "Carlos"]
filas = [{"Nombre": nombre, "Edad": 20 + i*5} for i, nombre in
enumerate(nombres)]

with open("archivo19.csv", "w", newline="") as archivo:
    campos = ["Nombre", "Edad"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()
    escritor.writerows(filas)
```

Ejercicio 66

Enunciado: Leer un CSV y mostrar los nombres de los mayores de 25 años.
Explicación: Filtramos por la columna "Edad" y mostramos la columna "Nombre".

```
import csv

with open("archivo19.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if int(fila["Edad"]) > 25:
            print(fila["Nombre"])
```

Ejercicio 67

Enunciado: Crear un CSV con empleados y su departamento usando lista de diccionarios.
Explicación: `DictWriter` permite escribir datos organizados con claves.

```
import csv

empleados = [
    {"Nombre": "Ana", "Departamento": "Ventas"},
    {"Nombre": "Luis", "Departamento": "Marketing"},
    {"Nombre": "Marta", "Departamento": "Finanzas"}
]

with open("archivo20.csv", "w", newline="") as archivo:
    campos = ["Nombre", "Departamento"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()
    escritor.writerows(empleados)
```

Ejercicio 68

Enunciado: Leer un CSV y mostrar los nombres de un departamento específico.
Explicación: Filtramos las filas comprobando el valor de la columna "Departamento".

```
import csv

departamento_buscado = "Marketing"
with open("archivo20.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if fila["Departamento"] == departamento_buscado:
            print(fila["Nombre"])
```

Ejercicio 69

Enunciado: Crear un CSV con productos y precios y guardarlo usando `writerows`.
Explicación: `writerows` permite escribir varias filas de una lista a la vez.

```
import csv

productos = [
    ["Producto", "Precio"],
    ["Manzana", 0.85],
    ["Banana", 1.25],
    ["Cereza", 2.50]
]

with open("archivo21.csv", "w", newline="") as archivo:
    escritor = csv.writer(archivo)
    escritor.writerows(productos)
```

Ejercicio 70

Enunciado: Leer un CSV y calcular el precio total de todos los productos.
Explicación: Sumamos los valores de la columna "Precio" para obtener el total.

```
import csv

total = 0
with open("archivo21.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        total += float(fila["Precio"])
print(f"Precio total: {total}")
```

Ejercicio 71

Enunciado: Filtrar un CSV y mostrar productos con precio mayor a 1.
Explicación: Comprobamos cada fila y mostramos solo las que cumplen la condición en la columna "Precio".

```
import csv

with open("archivo21.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if float(fila["Precio"]) > 1:
            print(fila)
```

Ejercicio 72

Enunciado: Leer un CSV y crear una lista de productos con precios menores a 2.
Explicación: Iteramos sobre las filas y agregamos a una lista los que cumplen la condición.

```
import csv

productos_baratos = []
with open("archivo21.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if float(fila["Precio"]) < 2:
            productos_baratos.append(fila["Producto"])
print(productos_baratos)
```

Ejercicio 73

Enunciado: Crear un CSV con alumnos y sus calificaciones usando `DictWriter`.
Explicación: Guardamos datos de manera organizada con claves correspondientes a los encabezados.

```
import csv

alumnos = [
    {"Nombre": "Ana", "Calificación": 8.5},
    {"Nombre": "Luis", "Calificación": 7.0},
    {"Nombre": "Marta", "Calificación": 9.2}
]

with open("archivo22.csv", "w", newline="") as archivo:
    campos = ["Nombre", "Calificación"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()
    escritor.writerows(alumnos)
```

Ejercicio 74

Enunciado: Leer un CSV y mostrar alumnos con calificación mayor a 8.
Explicación: Filtramos según la columna "Calificación".

```
import csv

with open("archivo22.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if float(fila["Calificación"]) > 8:
            print(fila)
```

Ejercicio 75

Enunciado: Crear un CSV con ciudades y poblaciones usando lista de diccionarios.
Explicación: Guardamos datos de manera estructurada usando `DictWriter`.

```
import csv

ciudades = [
    {"Ciudad": "Madrid", "Poblacion": 3200000},
    {"Ciudad": "Barcelona", "Poblacion": 1600000},
    {"Ciudad": "Valencia", "Poblacion": 800000}
]

with open("archivo23.csv", "w", newline="") as archivo:
    campos = ["Ciudad", "Poblacion"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()
    escritor.writerows(ciudades)
```

Ejercicio 76

Enunciado: Leer un CSV y mostrar la ciudad con mayor población.
Explicación: Iteramos comparando los valores de la columna "Poblacion" para encontrar el máximo.

```
import csv

max_ciudad = None
with open("archivo23.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if max_ciudad is None or int(fila["Poblacion"]) >
int(max_ciudad["Poblacion"]):
            max_ciudad = fila
print(max_ciudad)
```

Ejercicio 77

Enunciado: Leer un CSV y mostrar la ciudad con menor población.
Explicación: Iteramos comparando los valores de la columna "Poblacion" para encontrar el mínimo.

```
import csv

min_ciudad = None
with open("archivo23.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if min_ciudad is None or int(fila["Poblacion"]) <
int(min_ciudad["Poblacion"]):
            min_ciudad = fila
print(min_ciudad)
```

Ejercicio 78

Enunciado: Crear un CSV con nombres y números de teléfono usando lista de diccionarios.
Explicación: `DictWriter` permite guardar datos estructurados con claves y encabezados.

```
import csv

contactos = [
    {"Nombre": "Ana", "Telefono": "123456789"},
    {"Nombre": "Luis", "Telefono": "987654321"},
    {"Nombre": "Marta", "Telefono": "555555555"}
]

with open("archivo24.csv", "w", newline="") as archivo:
    campos = ["Nombre", "Telefono"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()
    escritor.writerows(contactos)
```

Ejercicio 79

Enunciado: Leer un CSV y mostrar los teléfonos de todos los contactos.
Explicación: Iteramos sobre las filas y mostramos la columna "Telefono".

```
import csv

with open("archivo24.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        print(fila["Telefono"])
```

Ejercicio 80

Enunciado: Crear un CSV con productos y stock y escribir varias filas con `writerows`.
Explicación: Guardamos varias filas de manera sencilla usando una lista de listas.

```
import csv

productos = [
    ["Producto", "Stock"],
    ["Manzana", 50],
    ["Banana", 30],
    ["Cereza", 20]
]

with open("archivo25.csv", "w", newline="") as archivo:
    escritor = csv.writer(archivo)
    escritor.writerows(productos)
```

Ejercicio 81

Enunciado: Leer un CSV y mostrar los productos con stock mayor a 25.
Explicación: Filtramos las filas comprobando el valor de la columna "Stock".

```
import csv

with open("archivo25.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if int(fila["Stock"]) > 25:
            print(fila)
```

Ejercicio 82

Enunciado: Leer un CSV y calcular el stock total de todos los productos.
Explicación: Sumamos los valores de la columna "Stock" para obtener el total.

```
import csv

total_stock = 0
with open("archivo25.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        total_stock += int(fila["Stock"])
print(f"Stock total: {total_stock}")
```

Ejercicio 83

Enunciado: Crear un CSV con alumnos y notas finales usando lista de diccionarios.
Explicación: `DictWriter` permite guardar los datos con encabezados claros.

```
import csv

alumnos = [
    {"Nombre": "Ana", "Nota": 9.0},
    {"Nombre": "Luis", "Nota": 7.5},
    {"Nombre": "Marta", "Nota": 8.3}
]

with open("archivo26.csv", "w", newline="") as archivo:
    campos = ["Nombre", "Nota"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()
    escritor.writerows(alumnos)
```

Ejercicio 84

Enunciado: Leer un CSV y mostrar los alumnos con nota menor a 8.
Explicación: Filtramos las filas según la columna "Nota".

```
import csv

with open("archivo26.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if float(fila["Nota"]) < 8:
            print(fila)
```

Ejercicio 85

Enunciado: Crear un CSV con ciudades y temperaturas usando lista de diccionarios.
Explicación: Guardamos datos de manera estructurada con encabezados y valores.

```
import csv

ciudades = [
    {"Ciudad": "Madrid", "Temperatura": 30},
    {"Ciudad": "Barcelona", "Temperatura": 28},
    {"Ciudad": "Valencia", "Temperatura": 32}
]

with open("archivo27.csv", "w", newline="") as archivo:
    campos = ["Ciudad", "Temperatura"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()
    escritor.writerows(ciudades)
```

Ejercicio 86

Enunciado: Leer un CSV y mostrar la ciudad con mayor temperatura.
Explicación: Comparamos los valores de la columna "Temperatura" para encontrar el máximo.

```
import csv

max_temp = None
with open("archivo27.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if max_temp is None or int(fila["Temperatura"]) >
int(max_temp["Temperatura"]):
            max_temp = fila
print(max_temp)
```

Ejercicio 87

Enunciado: Leer un CSV y mostrar la ciudad con menor temperatura.
Explicación: Comparamos los valores de la columna "Temperatura" para encontrar el mínimo.

```
import csv

min_temp = None
with open("archivo27.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if min_temp is None or int(fila["Temperatura"]) <
int(min_temp["Temperatura"]):
            min_temp = fila
print(min_temp)
```

Ejercicio 88

Enunciado: Crear un CSV con películas y puntuaciones usando lista de diccionarios.
Explicación: `DictWriter` permite almacenar datos organizados con claves.

```
import csv

peliculas = [
    {"Película": "Inception", "Puntuación": 9},
    {"Película": "Titanic", "Puntuación": 8},
    {"Película": "Avatar", "Puntuación": 7}
]

with open("archivo28.csv", "w", newline="") as archivo:
    campos = ["Película", "Puntuación"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()
    escritor.writerows(peliculas)
```

Ejercicio 89

Enunciado: Leer un CSV y mostrar las películas con puntuación mayor o igual a 8.
Explicación: Filtramos según la columna "Puntuación".

```
import csv

with open("archivo28.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if int(fila["Puntuación"]) >= 8:
            print(fila)
```

Ejercicio 90

Enunciado: Leer un CSV y calcular la puntuación promedio de todas las películas.
Explicación: Sumamos las puntuaciones y dividimos entre el número de filas.

```
import csv

total = 0
contador = 0
with open("archivo28.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        total += int(fila["Puntuación"])
        contador += 1
promedio = total / contador
print(f"Puntuación promedio: {promedio}")
```

Ejercicio 91

Enunciado: Crear un CSV con estudiantes y sus materias usando lista de diccionarios.
Explicación: `DictWriter` permite guardar datos organizados con claves que representan columnas.

```
import csv

estudiantes = [
    {"Nombre": "Ana", "Materia": "Matemáticas"},
    {"Nombre": "Luis", "Materia": "Historia"},
    {"Nombre": "Marta", "Materia": "Física"}
]

with open("archivo29.csv", "w", newline="") as archivo:
    campos = ["Nombre", "Materia"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()
    escritor.writerows(estudiantes)
```

Ejercicio 92

Enunciado: Leer un CSV y mostrar los estudiantes que cursan "Matemáticas".
Explicación: Filtramos según la columna "Materia".

```
import csv

with open("archivo29.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if fila["Materia"] == "Matemáticas":
            print(fila)
```

Ejercicio 93

Enunciado: Crear un CSV con clientes y sus compras usando `DictWriter`.
Explicación: Guardamos datos de clientes de forma estructurada para futuras operaciones.

```
import csv

clientes = [
    {"Nombre": "Ana", "Compra": 150},
    {"Nombre": "Luis", "Compra": 200},
    {"Nombre": "Marta", "Compra": 120}
]

with open("archivo30.csv", "w", newline="") as archivo:
    campos = ["Nombre", "Compra"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()
    escritor.writerows(clientes)
```

Ejercicio 94

Enunciado: Leer un CSV y mostrar los clientes con compra mayor a 150.
Explicación: Filtramos filas según la columna "Compra".

```
import csv

with open("archivo30.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if int(fila["Compra"]) > 150:
            print(fila)
```

Ejercicio 95

Enunciado: Crear un CSV con países y sus habitantes usando lista de diccionarios.
Explicación: Guardamos datos demográficos de manera organizada.

```
import csv

paises = [
    {"País": "España", "Habitantes": 47000000},
    {"País": "Francia", "Habitantes": 67000000},
    {"País": "Italia", "Habitantes": 60000000}
]

with open("archivo31.csv", "w", newline="") as archivo:
    campos = ["País", "Habitantes"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()
    escritor.writerows(paises)
```

Ejercicio 96

Enunciado: Leer un CSV y mostrar el país con mayor población.
Explicación: Iteramos comparando la columna "Habitantes" para encontrar el máximo.

```
import csv

max_pais = None
with open("archivo31.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if max_pais is None or int(fila["Habitantes"]) >
int(max_pais["Habitantes"]):
            max_pais = fila
print(max_pais)
```

Ejercicio 97

Enunciado: Leer un CSV y mostrar el país con menor población.
Explicación: Iteramos comparando la columna "Habitantes" para encontrar el mínimo.

```
import csv

min_pais = None
with open("archivo31.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if min_pais is None or int(fila["Habitantes"]) <
int(min_pais["Habitantes"]):
            min_pais = fila
print(min_pais)
```

Ejercicio 98

Enunciado: Crear un CSV con libros y precios usando `DictWriter`.

Explicación: Guardamos datos de libros y precios de manera clara y estructurada.

```
import csv

libros = [
    {"Libro": "Python Básico", "Precio": 25},
    {"Libro": "Aprende CSV", "Precio": 30},
    {"Libro": "Programación Fácil", "Precio": 20}
]

with open("archivo32.csv", "w", newline="") as archivo:
    campos = ["Libro", "Precio"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()
    escritor.writerows(libros)
```

Ejercicio 99

Enunciado: Leer un CSV y mostrar los libros con precio mayor a 20.

Explicación: Filtramos filas según la columna "Precio".

```
import csv

with open("archivo32.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        if int(fila["Precio"]) > 20:
            print(fila)
```

Ejercicio 100

Enunciado: Leer un CSV y calcular el precio promedio de los libros.

Explicación: Sumamos los precios y los dividimos entre el número de filas para obtener el promedio.

```
import csv

total = 0
contador = 0
with open("archivo32.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        total += int(fila["Precio"])
        contador += 1
promedio = total / contador
print(f"Precio promedio de libros: {promedio}")
```

Ejercicios python archivos json

Ejercicio 1

Enunciado: Crear un archivo JSON con un diccionario simple y guardarlo.
Explicación: Guardamos un diccionario en un archivo JSON usando `json.dump()`.

```
import json

data = {"nombre": "Ana", "edad": 25}
with open("archivo1.json", "w") as archivo:
    json.dump(data, archivo)
print("Archivo JSON creado.")
```

Ejercicio 2

Enunciado: Leer un archivo JSON y mostrar su contenido.
Explicación: Usamos `json.load()` para cargar el diccionario desde el archivo.

```
import json

with open("archivo1.json", "r") as archivo:
    data = json.load(archivo)
print(data)
```

Ejercicio 3

Enunciado: Crear un JSON con una lista de números y guardarlo.
Explicación: JSON puede guardar listas, no solo diccionarios.

```
import json

numeros = [1, 2, 3, 4, 5]
with open("archivo2.json", "w") as archivo:
    json.dump(numeros, archivo)
print("Archivo JSON con lista creado.")
```

Ejercicio 4

Enunciado: Leer un JSON que contiene una lista y mostrar los números.
Explicación: Cargamos la lista con `json.load()` y la imprimimos.

```
import json

with open("archivo2.json", "r") as archivo:
    numeros = json.load(archivo)
print(numeros)
```

Ejercicio 5

Enunciado: Crear un JSON con un diccionario de productos y precios.
Explicación: Guardamos un diccionario anidado con varios pares clave-valor.

```
import json

productos = {"manzana": 0.85, "banana": 1.25, "cereza": 2.5}
with open("archivo3.json", "w") as archivo:
    json.dump(productos, archivo)
print("Archivo JSON de productos creado.")
```

Ejercicio 6

Enunciado: Leer un JSON y mostrar el precio de la "banana".
Explicación: Accedemos al valor usando la clave del diccionario.

```
import json

with open("archivo3.json", "r") as archivo:
    productos = json.load(archivo)
print("Precio de la banana:", productos["banana"])
```

Ejercicio 7

Enunciado: Crear un JSON con una lista de diccionarios (empleados).
Explicación: JSON puede contener estructuras complejas, como listas de diccionarios.

```
import json

empleados = [
    {"nombre": "Ana", "edad": 25},
    {"nombre": "Luis", "edad": 30},
    {"nombre": "Marta", "edad": 28}
]
with open("archivo4.json", "w") as archivo:
    json.dump(empleados, archivo)
print("Archivo JSON con lista de diccionarios creado.")
```

Ejercicio 8

Enunciado: Leer un JSON con lista de empleados y mostrar los nombres.
Explicación: Iteramos sobre la lista de diccionarios y accedemos a la clave "nombre".

```
import json

with open("archivo4.json", "r") as archivo:
    empleados = json.load(archivo)
for empleado in empleados:
    print(empleado["nombre"])
```

Ejercicio 9

Enunciado: Crear un JSON con información anidada (persona con dirección).
Explicación: JSON permite diccionarios dentro de diccionarios para datos jerárquicos.

```
import json

persona = {
    "nombre": "Ana",
    "edad": 25,
    "direccion": {"calle": "Calle Mayor", "ciudad": "Madrid"}
}
with open("archivo5.json", "w") as archivo:
    json.dump(persona, archivo)
print("Archivo JSON con datos anidados creado.")
```

Ejercicio 10

Enunciado: Leer un JSON con datos anidados y mostrar la ciudad de la persona.
Explicación: Accedemos a la clave anidada dentro del diccionario.

```
import json

with open("archivo5.json", "r") as archivo:
    persona = json.load(archivo)
print("Ciudad:", persona["direccion"]["ciudad"])
```

Ejercicio 11

Enunciado: Crear un JSON con calificaciones de alumnos y guardarlo.
Explicación: Guardamos un diccionario con nombres de alumnos como claves y sus notas como valores.

```
import json

calificaciones = {"Ana": 8.5, "Luis": 7.0, "Marta": 9.2}
with open("archivo6.json", "w") as archivo:
    json.dump(calificaciones, archivo)
print("Archivo JSON con calificaciones creado.")
```

Ejercicio 12

Enunciado: Leer un JSON y mostrar la calificación de "Marta".
Explicación: Accedemos a la clave correspondiente del diccionario.

```
import json

with open("archivo6.json", "r") as archivo:
    calificaciones = json.load(archivo)
print("Calificación de Marta:", calificaciones["Marta"])
```

Ejercicio 13

Enunciado: Crear un JSON con una lista de ciudades.
Explicación: JSON puede almacenar listas simples de elementos.

```
import json

ciudades = ["Madrid", "Barcelona", "Valencia"]
with open("archivo7.json", "w") as archivo:
    json.dump(ciudades, archivo)
print("Archivo JSON con lista de ciudades creado.")
```

Ejercicio 14

Enunciado: Leer un JSON con lista de ciudades y mostrarlas en pantalla.
Explicación: Cargamos la lista y usamos un bucle para imprimir cada ciudad.

```
import json

with open("archivo7.json", "r") as archivo:
    ciudades = json.load(archivo)
for ciudad in ciudades:
    print(ciudad)
```

Ejercicio 15

Enunciado: Crear un JSON con información de productos (nombre y precio).
Explicación: Guardamos una lista de diccionarios para representar cada producto y su precio.

```
import json

productos = [
    {"nombre": "Manzana", "precio": 0.85},
    {"nombre": "Banana", "precio": 1.25},
    {"nombre": "Cereza", "precio": 2.5}
]
with open("archivo8.json", "w") as archivo:
    json.dump(productos, archivo)
print("Archivo JSON con productos creado.")
```

Ejercicio 16

Enunciado: Leer un JSON y mostrar los nombres de los productos.
Explicación: Iteramos sobre la lista de diccionarios y accedemos a la clave "nombre".

```
import json

with open("archivo8.json", "r") as archivo:
    productos = json.load(archivo)
for producto in productos:
    print(producto["nombre"])
```

Ejercicio 17

Enunciado: Crear un JSON con empleados y sus departamentos.
Explicación: Guardamos datos de empleados usando lista de diccionarios.

```
import json

empleados = [
    {"nombre": "Ana", "departamento": "Ventas"},
    {"nombre": "Luis", "departamento": "Marketing"},
    {"nombre": "Marta", "departamento": "Finanzas"}
]
with open("archivo9.json", "w") as archivo:
    json.dump(empleados, archivo)
print("Archivo JSON con empleados creado.")
```

Ejercicio 18

Enunciado: Leer un JSON y mostrar los empleados del departamento "Marketing".
Explicación: Filtramos la lista de diccionarios comprobando la clave "departamento".

```
import json

with open("archivo9.json", "r") as archivo:
    empleados = json.load(archivo)
for empleado in empleados:
    if empleado["departamento"] == "Marketing":
        print(empleado)
```

Ejercicio 19

Enunciado: Crear un JSON con estudiantes y sus materias usando datos anidados.
Explicación: Cada estudiante tiene un diccionario con nombre y lista de materias.

```
import json

estudiantes = [
    {"nombre": "Ana", "materias": ["Matemáticas", "Física"]},
    {"nombre": "Luis", "materias": ["Historia", "Geografía"]},
    {"nombre": "Marta", "materias": ["Biología", "Química"]}
]
with open("archivo10.json", "w") as archivo:
    json.dump(estudiantes, archivo)
print("Archivo JSON con estudiantes y materias creado.")
```

Ejercicio 20

Enunciado: Leer un JSON y mostrar las materias de "Ana".
Explicación: Iteramos sobre la lista de diccionarios y verificamos la clave "nombre".

```
import json

with open("archivo10.json", "r") as archivo:
    estudiantes = json.load(archivo)
for estudiante in estudiantes:
    if estudiante["nombre"] == "Ana":
        print("Materias de Ana:", estudiante["materias"])
```

Ejercicio 21

Enunciado: Crear un JSON con información de libros (título y autor).
Explicación: Guardamos una lista de diccionarios, cada uno representando un libro.

```
import json

libros = [
    {"titulo": "Python Básico", "autor": "Juan"},
    {"titulo": "Aprende JSON", "autor": "Ana"},
    {"titulo": "Programación Fácil", "autor": "Luis"}
]
with open("archivo11.json", "w") as archivo:
    json.dump(libros, archivo)
print("Archivo JSON con libros creado.")
```

Ejercicio 22

Enunciado: Leer un JSON y mostrar los títulos de los libros.
Explicación: Iteramos sobre la lista de diccionarios y accedemos a la clave "titulo".

```
import json

with open("archivo11.json", "r") as archivo:
    libros = json.load(archivo)
for libro in libros:
    print(libro["titulo"])
```

Ejercicio 23

Enunciado: Crear un JSON con ciudades y sus poblaciones.
Explicación: Guardamos un diccionario con nombres de ciudades como claves y población como valores.

```
import json

ciudades = {"Madrid": 3200000, "Barcelona": 1600000, "Valencia":
800000}
with open("archivo12.json", "w") as archivo:
    json.dump(ciudades, archivo)
print("Archivo JSON con ciudades creado.")
```

Ejercicio 24

Enunciado: Leer un JSON y mostrar la población de "Barcelona".
Explicación: Accedemos al valor del diccionario usando la clave correspondiente.

```
import json

with open("archivo12.json", "r") as archivo:
    ciudades = json.load(archivo)
print("Población de Barcelona:", ciudades["Barcelona"])
```

Ejercicio 25

Enunciado: Crear un JSON con empleados y edades.
Explicación: Guardamos una lista de diccionarios representando cada empleado.

```
import json

empleados = [
    {"nombre": "Ana", "edad": 25},
    {"nombre": "Luis", "edad": 30},
    {"nombre": "Marta", "edad": 28}
]
with open("archivo13.json", "w") as archivo:
    json.dump(empleados, archivo)
print("Archivo JSON con empleados creado.")
```

Ejercicio 26

Enunciado: Leer un JSON y mostrar los empleados mayores de 28 años.
Explicación: Filtramos la lista comprobando la clave "edad".

```
import json

with open("archivo13.json", "r") as archivo:
    empleados = json.load(archivo)
for empleado in empleados:
    if empleado["edad"] > 28:
        print(empleado)
```

Ejercicio 27

Enunciado: Crear un JSON con productos y stock disponible.
Explicación: Guardamos un diccionario con producto como clave y stock como valor.

```
import json

productos = {"manzana": 50, "banana": 30, "cereza": 20}
with open("archivo14.json", "w") as archivo:
    json.dump(productos, archivo)
print("Archivo JSON con productos creado.")
```

Ejercicio 28

Enunciado: Leer un JSON y mostrar los productos con stock menor a 40.
Explicación: Iteramos sobre el diccionario y aplicamos la condición.

```
import json

with open("archivo14.json", "r") as archivo:
    productos = json.load(archivo)
for producto, stock in productos.items():
    if stock < 40:
        print(producto, "tiene stock menor a 40:", stock)
```

Ejercicio 29

Enunciado: Crear un JSON con estudiantes y sus notas en varias materias.
Explicación: Cada estudiante es un diccionario con nombre y otra lista/diccionario de notas.

```
import json

estudiantes = [
    {"nombre": "Ana", "notas": {"Matemáticas": 9, "Física": 8}},
    {"nombre": "Luis", "notas": {"Historia": 7, "Geografía": 8}},
    {"nombre": "Marta", "notas": {"Biología": 10, "Química": 9}}
]
with open("archivo15.json", "w") as archivo:
    json.dump(estudiantes, archivo)
print("Archivo JSON con estudiantes y notas creado.")
```

Ejercicio 30

Enunciado: Leer un JSON y mostrar la nota de "Ana" en Matemáticas.
Explicación: Accedemos a la lista y al diccionario anidado correspondiente.

```
import json

with open("archivo15.json", "r") as archivo:
    estudiantes = json.load(archivo)
for estudiante in estudiantes:
    if estudiante["nombre"] == "Ana":
        print("Nota de Ana en Matemáticas:", estudiante["notas"]
["Matemáticas"])
```

Ejercicio 31

Enunciado: Crear un JSON con información de películas (título y director).
Explicación: Guardamos una lista de diccionarios, cada uno representando una película.

```
import json

peliculas = [
    {"titulo": "Inception", "director": "Nolan"},
    {"titulo": "Titanic", "director": "Cameron"},
    {"titulo": "Avatar", "director": "Cameron"}
]
with open("archivo16.json", "w") as archivo:
    json.dump(peliculas, archivo)
print("Archivo JSON con películas creado.")
```

Ejercicio 32

Enunciado: Leer un JSON y mostrar los títulos de las películas.
Explicación: Iteramos sobre la lista de diccionarios y accedemos a la clave "titulo".

```
import json

with open("archivo16.json", "r") as archivo:
    peliculas = json.load(archivo)
for pelicula in peliculas:
    print(pelicula["titulo"])
```

Ejercicio 33

Enunciado: Crear un JSON con un diccionario de países y sus capitales.
Explicación: Guardamos un diccionario simple con país como clave y capital como valor.

```
import json

paises = {"España": "Madrid", "Francia": "París", "Italia":
"Roma"}
with open("archivo17.json", "w") as archivo:
    json.dump(paises, archivo)
print("Archivo JSON con países y capitales creado.")
```

Ejercicio 34

Enunciado: Leer un JSON y mostrar la capital de "Italia".
Explicación: Accedemos al valor del diccionario usando la clave correspondiente.

```
import json

with open("archivo17.json", "r") as archivo:
    paises = json.load(archivo)
print("Capital de Italia:", paises["Italia"])
```

Ejercicio 35

Enunciado: Crear un JSON con empleados y sus salarios.
Explicación: Guardamos una lista de diccionarios representando cada empleado y su salario.

```
import json

empleados = [
    {"nombre": "Ana", "salario": 2500},
    {"nombre": "Luis", "salario": 3000},
    {"nombre": "Marta", "salario": 2800}
]
with open("archivo18.json", "w") as archivo:
    json.dump(empleados, archivo)
print("Archivo JSON con empleados y salarios creado.")
```

Ejercicio 36

Enunciado: Leer un JSON y mostrar los empleados con salario mayor a 2700.
Explicación: Filtramos la lista comprobando la clave "salario".

```
import json

with open("archivo18.json", "r") as archivo:
    empleados = json.load(archivo)
for empleado in empleados:
    if empleado["salario"] > 2700:
        print(empleado)
```

Ejercicio 37

Enunciado: Crear un JSON con productos y sus categorías.
Explicación: Guardamos una lista de diccionarios, cada producto con su categoría.

```
import json

productos = [
    {"producto": "Manzana", "categoria": "Fruta"},
    {"producto": "Leche", "categoria": "Lácteo"},
    {"producto": "Pan", "categoria": "Panadería"}
]
with open("archivo19.json", "w") as archivo:
    json.dump(productos, archivo)
print("Archivo JSON con productos y categorías creado.")
```

Ejercicio 38

Enunciado: Leer un JSON y mostrar todos los productos de la categoría "Fruta".
Explicación: Filtramos la lista comprobando la clave "categoria".

```
import json

with open("archivo19.json", "r") as archivo:
    productos = json.load(archivo)
for producto in productos:
    if producto["categoria"] == "Fruta":
        print(producto)
```

Ejercicio 39

Enunciado: Crear un JSON con estudiantes y su información académica (edad y promedio).
Explicación: Cada estudiante es un diccionario con datos básicos y un promedio.

```
import json

estudiantes = [
    {"nombre": "Ana", "edad": 20, "promedio": 8.5},
    {"nombre": "Luis", "edad": 21, "promedio": 7.0},
    {"nombre": "Marta", "edad": 22, "promedio": 9.2}
]
with open("archivo20.json", "w") as archivo:
    json.dump(estudiantes, archivo)
print("Archivo JSON con estudiantes creado.")
```

Ejercicio 40

Enunciado: Leer un JSON y mostrar el promedio de "Luis".
Explicación: Iteramos sobre la lista de diccionarios y accedemos al valor de "promedio" según el nombre.

```
import json

with open("archivo20.json", "r") as archivo:
    estudiantes = json.load(archivo)
for estudiante in estudiantes:
    if estudiante["nombre"] == "Luis":
        print("Promedio de Luis:", estudiante["promedio"])
```

Ejercicio 41

Enunciado: Crear un JSON con clientes y sus compras totales.
Explicación: Guardamos una lista de diccionarios, cada cliente con su monto total.

```
import json

clientes = [
    {"nombre": "Ana", "compra": 150},
    {"nombre": "Luis", "compra": 200},
    {"nombre": "Marta", "compra": 120}
]
with open("archivo21.json", "w") as archivo:
    json.dump(clientes, archivo)
print("Archivo JSON con clientes creado.")
```

Ejercicio 42

Enunciado: Leer un JSON y mostrar los clientes con compra mayor a 150.
Explicación: Filtramos la lista comprobando la clave "compra".

```
import json

with open("archivo21.json", "r") as archivo:
    clientes = json.load(archivo)
for cliente in clientes:
    if cliente["compra"] > 150:
        print(cliente)
```

Ejercicio 43

Enunciado: Crear un JSON con ciudades y su temperatura promedio.
Explicación: Guardamos un diccionario con ciudad como clave y temperatura como valor.

```
import json

temperaturas = {"Madrid": 30, "Barcelona": 28, "Valencia": 32}
with open("archivo22.json", "w") as archivo:
    json.dump(temperaturas, archivo)
print("Archivo JSON con temperaturas creado.")
```

Ejercicio 44

Enunciado: Leer un JSON y mostrar la ciudad con mayor temperatura.
Explicación: Iteramos sobre el diccionario para encontrar el valor máximo.

```
import json

with open("archivo22.json", "r") as archivo:
    temperaturas = json.load(archivo)
ciudad_max = max(temperaturas, key=temperaturas.get)
print("Ciudad con mayor temperatura:", ciudad_max,
temperaturas[ciudad_max])
```

Ejercicio 45

Enunciado: Crear un JSON con cursos y sus inscritos.
Explicación: Guardamos un diccionario con curso como clave y número de inscritos como valor.

```
import json

cursos = {"Python": 25, "Java": 30, "C#": 20}
with open("archivo23.json", "w") as archivo:
    json.dump(cursos, archivo)
print("Archivo JSON con cursos creado.")
```

Ejercicio 46

Enunciado: Leer un JSON y mostrar los cursos con más de 25 inscritos.
Explicación: Iteramos sobre el diccionario y aplicamos la condición sobre el valor.

```
import json

with open("archivo23.json", "r") as archivo:
    cursos = json.load(archivo)
for curso, inscritos in cursos.items():
    if inscritos > 25:
        print(curso, "tiene más de 25 inscritos:", inscritos)
```

Ejercicio 47

Enunciado: Crear un JSON con productos y stock disponible en varias tiendas.
Explicación: Cada producto tiene un diccionario con tienda y cantidad.

```
import json

productos = {
    "manzana": {"Tienda1": 50, "Tienda2": 30},
    "banana": {"Tienda1": 40, "Tienda2": 25},
    "cereza": {"Tienda1": 20, "Tienda2": 15}
}
with open("archivo24.json", "w") as archivo:
    json.dump(productos, archivo)
print("Archivo JSON con productos por tienda creado.")
```

Ejercicio 48

Enunciado: Leer un JSON y mostrar el stock de "manzana" en "Tienda2".
Explicación: Accedemos a las claves anidadas para obtener el valor requerido.

```
import json

with open("archivo24.json", "r") as archivo:
    productos = json.load(archivo)
print("Stock de manzana en Tienda2:", productos["manzana"]
["Tienda2"])
```

Ejercicio 49

Enunciado: Crear un JSON con jugadores y sus estadísticas (puntos y rebotes).
Explicación: Cada jugador es un diccionario con valores numéricos de estadísticas.

```
import json

jugadores = [
    {"nombre": "Juan", "puntos": 25, "rebotes": 10},
    {"nombre": "Luis", "puntos": 20, "rebotes": 12},
    {"nombre": "Marta", "puntos": 30, "rebotes": 8}
]
with open("archivo25.json", "w") as archivo:
    json.dump(jugadores, archivo)
print("Archivo JSON con jugadores creado.")
```

Ejercicio 50

Enunciado: Leer un JSON y mostrar los jugadores con más de 20 puntos.
Explicación: Filtramos la lista de diccionarios según la clave "puntos".

```
import json

with open("archivo25.json", "r") as archivo:
    jugadores = json.load(archivo)
for jugador in jugadores:
    if jugador["puntos"] > 20:
        print(jugador)
```

Si quieres, puedo continuar con el **Bloque 6 (ejercicios 51 al 60)** para seguir avanzando hacia los 100 programas de JSON.

¿Deseas que continúe con el **Bloque 6**?

¡Perfecto! 🌟 Aquí tienes el **Bloque 6 de 10 programas (51 al 60)** sobre **archivos JSON**, nivel fácil, con enunciado, explicación y código:

Ejercicio 51

Enunciado: Crear un JSON con canciones y sus duraciones en minutos.
Explicación: Cada canción es un diccionario con título y duración; guardamos todo en una lista.

```
import json

canciones = [
    {"titulo": "Canción1", "duracion": 3.5},
    {"titulo": "Canción2", "duracion": 4.2},
    {"titulo": "Canción3", "duracion": 5.0}
]
with open("archivo26.json", "w") as archivo:
    json.dump(canciones, archivo)
print("Archivo JSON con canciones creado.")
```

Ejercicio 52

Enunciado: Leer un JSON y mostrar las canciones que duren más de 4 minutos.
Explicación: Iteramos sobre la lista y filtramos según la clave "duracion".

```
import json

with open("archivo26.json", "r") as archivo:
    canciones = json.load(archivo)
for cancion in canciones:
    if cancion["duracion"] > 4:
        print(cancion)
```

Ejercicio 53

Enunciado: Crear un JSON con países y su moneda.
Explicación: Guardamos un diccionario simple con país como clave y moneda como valor.

```
import json

monedas = {"España": "Euro", "USA": "Dólar", "Japón": "Yen"}
with open("archivo27.json", "w") as archivo:
    json.dump(monedas, archivo)
print("Archivo JSON con monedas creado.")
```

Ejercicio 54

Enunciado: Leer un JSON y mostrar la moneda de "USA".
Explicación: Accedemos al diccionario usando la clave correspondiente.

```
import json

with open("archivo27.json", "r") as archivo:
    monedas = json.load(archivo)
print("Moneda de USA:", monedas["USA"])
```

Ejercicio 55

Enunciado: Crear un JSON con autos y sus características (color y año).
Explicación: Cada auto es un diccionario con valores descriptivos; guardamos todo en una lista.

```
import json

autos = [
    {"marca": "Toyota", "color": "Rojo", "año": 2020},
    {"marca": "Ford", "color": "Azul", "año": 2018},
    {"marca": "Honda", "color": "Blanco", "año": 2021}
]
with open("archivo28.json", "w") as archivo:
    json.dump(autos, archivo)
print("Archivo JSON con autos creado.")
```

Ejercicio 56

Enunciado: Leer un JSON y mostrar los autos del año 2020 o superior.
Explicación: Iteramos sobre la lista de diccionarios y filtramos según la clave "año".

```
import json

with open("archivo28.json", "r") as archivo:
    autos = json.load(archivo)
for auto in autos:
    if auto["año"] >= 2020:
        print(auto)
```

Ejercicio 57

Enunciado: Crear un JSON con jugadores y su posición en un equipo.
Explicación: Cada jugador es un diccionario con nombre y posición; guardamos una lista de jugadores.

```
import json

jugadores = [
    {"nombre": "Juan", "posicion": "Delantero"},
    {"nombre": "Luis", "posicion": "Portero"},
    {"nombre": "Marta", "posicion": "Defensa"}
]
with open("archivo29.json", "w") as archivo:
    json.dump(jugadores, archivo)
print("Archivo JSON con jugadores creado.")
```

Ejercicio 58

Enunciado: Leer un JSON y mostrar todos los jugadores que sean delanteros.
Explicación: Filtramos la lista de diccionarios según la clave "posicion".

```
import json

with open("archivo29.json", "r") as archivo:
    jugadores = json.load(archivo)
for jugador in jugadores:
    if jugador["posicion"] == "Delantero":
        print(jugador)
```

Ejercicio 59

Enunciado: Crear un JSON con ciudades y sus coordenadas (latitud y longitud).
Explicación: Cada ciudad tiene un diccionario con coordenadas anidadas.

```
import json

ciudades = {
    "Madrid": {"lat": 40.4168, "lon": -3.7038},
    "Barcelona": {"lat": 41.3851, "lon": 2.1734},
    "Valencia": {"lat": 39.4699, "lon": -0.3763}
}
with open("archivo30.json", "w") as archivo:
    json.dump(ciudades, archivo)
print("Archivo JSON con ciudades y coordenadas creado.")
```

Ejercicio 60

Enunciado: Leer un JSON y mostrar la latitud de "Barcelona".
Explicación: Accedemos a las claves anidadas para obtener el valor deseado.

```
import json

with open("archivo30.json", "r") as archivo:
    ciudades = json.load(archivo)
print("Latitud de Barcelona:", ciudades["Barcelona"]["lat"])
```

Ejercicio 61

Enunciado: Crear un JSON con cursos y lista de alumnos inscritos en cada curso.
Explicación: Guardamos un diccionario con curso como clave y lista de alumnos como valor.

```
import json

cursos = {
    "Python": ["Ana", "Luis", "Marta"],
    "Java": ["Juan", "Carlos", "Laura"],
    "C#": ["Marta", "Luis", "Ana"]
}
with open("archivo31.json", "w") as archivo:
    json.dump(cursos, archivo)
print("Archivo JSON con cursos y alumnos creado.")
```

Ejercicio 62

Enunciado: Leer un JSON y mostrar los alumnos del curso "Python".
Explicación: Accedemos al valor del diccionario según la clave del curso.

```
import json

with open("archivo31.json", "r") as archivo:
    cursos = json.load(archivo)
print("Alumnos del curso Python:", cursos["Python"])
```

Ejercicio 63

Enunciado: Crear un JSON con películas y sus géneros.
Explicación: Guardamos un diccionario donde cada película tiene como valor su género.

```
import json

peliculas = {
    "Inception": "Ciencia Ficción",
    "Titanic": "Romance",
    "Avatar": "Aventura"
}
with open("archivo32.json", "w") as archivo:
    json.dump(peliculas, archivo)
print("Archivo JSON con películas y géneros creado.")
```

Ejercicio 64

Enunciado: Leer un JSON y mostrar las películas de género "Romance".
Explicación: Iteramos sobre el diccionario y verificamos el valor para filtrar.

```
import json

with open("archivo32.json", "r") as archivo:
    peliculas = json.load(archivo)
for pelicula, genero in peliculas.items():
    if genero == "Romance":
        print(pelicula)
```

Ejercicio 65

Enunciado: Crear un JSON con estudiantes y sus habilidades (lista de strings).
Explicación: Cada estudiante tiene una lista de habilidades; guardamos todo en una lista de diccionarios.

```
import json

estudiantes = [
    {"nombre": "Ana", "habilidades": ["Python", "Excel"]},
    {"nombre": "Luis", "habilidades": ["Java", "SQL"]},
    {"nombre": "Marta", "habilidades": ["C#", "Python"]}
]
with open("archivo33.json", "w") as archivo:
    json.dump(estudiantes, archivo)
print("Archivo JSON con estudiantes y habilidades creado.")
```

Ejercicio 66

Enunciado: Leer un JSON y mostrar los estudiantes que saben "Python".
Explicación: Iteramos sobre la lista de diccionarios y comprobamos si la lista de habilidades contiene "Python".

```
import json

with open("archivo33.json", "r") as archivo:
    estudiantes = json.load(archivo)
for estudiante in estudiantes:
    if "Python" in estudiante["habilidades"]:
        print(estudiante["nombre"])
```

Ejercicio 67

Enunciado: Crear un JSON con ciudades y su información detallada (población y país).
Explicación: Cada ciudad tiene un diccionario con valores de población y país.

```
import json

ciudades = {
    "Madrid": {"poblacion": 3200000, "pais": "España"},
    "Paris": {"poblacion": 2140000, "pais": "Francia"},
    "Roma": {"poblacion": 2870000, "pais": "Italia"}
}
with open("archivo34.json", "w") as archivo:
    json.dump(ciudades, archivo)
print("Archivo JSON con ciudades detalladas creado.")
```

Ejercicio 68

Enunciado: Leer un JSON y mostrar la población de "Paris".
Explicación: Accedemos a las claves anidadas para obtener el valor de población.

```
import json

with open("archivo34.json", "r") as archivo:
    ciudades = json.load(archivo)
print("Población de Paris:", ciudades["Paris"]["poblacion"])
```

Ejercicio 69

Enunciado: Crear un JSON con equipos deportivos y sus jugadores.
Explicación: Cada equipo tiene una lista de jugadores; guardamos todo en un diccionario.

```
import json

equipos = {
    "Barcelona": ["Messi", "Xavi", "Iniesta"],
    "Real Madrid": ["Ronaldo", "Benzema", "Modric"]
}
with open("archivo35.json", "w") as archivo:
    json.dump(equipos, archivo)
print("Archivo JSON con equipos y jugadores creado.")
```

Ejercicio 70

Enunciado: Leer un JSON y mostrar los jugadores del equipo "Barcelona".
Explicación: Accedemos al valor de la clave correspondiente en el diccionario.

```
import json

with open("archivo35.json", "r") as archivo:
    equipos = json.load(archivo)
print("Jugadores de Barcelona:", equipos["Barcelona"])
```

Si quieres, puedo continuar con el **Bloque 8 (ejercicios 71 al 80)** para seguir avanzando hacia los 100 programas de JSON.

¿Deseas que continúe con el **Bloque 8**?

¡Perfecto! 🌟 Aquí tienes el **Bloque 8 de 10 programas (71 al 80)** sobre **archivos JSON**, nivel fácil, con enunciado, explicación y código:

Ejercicio 71

Enunciado: Crear un JSON con artistas y sus géneros musicales.
Explicación: Guardamos un diccionario donde cada artista tiene como valor su género musical.

```
import json

artistas = {
    "Adele": "Pop",
    "Metallica": "Rock",
    "Beethoven": "Clásica"
}
with open("archivo36.json", "w") as archivo:
    json.dump(artistas, archivo)
print("Archivo JSON con artistas y géneros creado.")
```

Ejercicio 72

Enunciado: Leer un JSON y mostrar los artistas de género "Rock".
Explicación: Iteramos sobre el diccionario y filtramos por valor.

```
import json

with open("archivo36.json", "r") as archivo:
    artistas = json.load(archivo)
for artista, genero in artistas.items():
    if genero == "Rock":
        print(artista)
```

Ejercicio 73

Enunciado: Crear un JSON con libros y su disponibilidad (True o False).
Explicación: Guardamos un diccionario con título como clave y disponibilidad como valor booleano.

```
import json

libros = {"Libro1": True, "Libro2": False, "Libro3": True}
with open("archivo37.json", "w") as archivo:
    json.dump(libros, archivo)
print("Archivo JSON con libros y disponibilidad creado.")
```

Ejercicio 74

Enunciado: Leer un JSON y mostrar los libros disponibles.
Explicación: Iteramos sobre el diccionario y filtramos por valor True.

```
import json

with open("archivo37.json", "r") as archivo:
    libros = json.load(archivo)
for libro, disponible in libros.items():
    if disponible:
        print(libro, "está disponible")
```

Ejercicio 75

Enunciado: Crear un JSON con alumnos y sus notas finales.
Explicación: Guardamos una lista de diccionarios, cada uno representando un alumno y su nota final.

```
import json

alumnos = [
    {"nombre": "Ana", "nota": 8.5},
    {"nombre": "Luis", "nota": 7.0},
    {"nombre": "Marta", "nota": 9.2}
]
with open("archivo38.json", "w") as archivo:
    json.dump(alumnos, archivo)
print("Archivo JSON con alumnos y notas creado.")
```

Ejercicio 76

Enunciado: Leer un JSON y mostrar los alumnos con nota mayor a 8.
Explicación: Iteramos sobre la lista de diccionarios y filtramos por la clave "nota".

```
import json

with open("archivo38.json", "r") as archivo:
    alumnos = json.load(archivo)
for alumno in alumnos:
    if alumno["nota"] > 8:
        print(alumno)
```

Ejercicio 77

Enunciado: Crear un JSON con empleados y sus departamentos.
Explicación: Guardamos una lista de diccionarios con nombre y departamento de cada empleado.

```
import json

empleados = [
    {"nombre": "Ana", "departamento": "Ventas"},
    {"nombre": "Luis", "departamento": "Marketing"},
    {"nombre": "Marta", "departamento": "Finanzas"}
]
with open("archivo39.json", "w") as archivo:
    json.dump(empleados, archivo)
print("Archivo JSON con empleados y departamentos creado.")
```

Ejercicio 78

Enunciado: Leer un JSON y mostrar los empleados del departamento "Ventas".
Explicación: Filtramos la lista de diccionarios comprobando la clave "departamento".

```
import json

with open("archivo39.json", "r") as archivo:
    empleados = json.load(archivo)
for empleado in empleados:
    if empleado["departamento"] == "Ventas":
        print(empleado)
```

Ejercicio 79

Enunciado: Crear un JSON con productos y sus precios, incluyendo IVA.
Explicación: Guardamos una lista de diccionarios, cada producto con precio base y precio con IVA.

```
import json

productos = [
    {"producto": "Manzana", "precio": 0.85, "precio_iva": 1.03},
    {"producto": "Leche", "precio": 1.2, "precio_iva": 1.45},
    {"producto": "Pan", "precio": 1.0, "precio_iva": 1.21}
]
with open("archivo40.json", "w") as archivo:
    json.dump(productos, archivo)
print("Archivo JSON con productos y precios creado.")
```

Ejercicio 80

Enunciado: Leer un JSON y mostrar los productos con precio con IVA mayor a 1.2.
Explicación: Iteramos sobre la lista de diccionarios y filtramos por la clave "precio_iva".

```
import json

with open("archivo40.json", "r") as archivo:
    productos = json.load(archivo)
for producto in productos:
    if producto["precio_iva"] > 1.2:
        print(producto)
```

Ejercicio 81

Enunciado: Crear un JSON con películas y su duración en minutos.
Explicación: Cada película es un diccionario con título y duración; guardamos todos en una lista.

```
import json

peliculas = [
    {"titulo": "Inception", "duracion": 148},
    {"titulo": "Titanic", "duracion": 195},
    {"titulo": "Avatar", "duracion": 162}
]
with open("archivo41.json", "w") as archivo:
    json.dump(peliculas, archivo)
print("Archivo JSON con películas y duración creado.")
```

Ejercicio 82

Enunciado: Leer un JSON y mostrar las películas que duren más de 160 minutos.
Explicación: Iteramos sobre la lista y filtramos por la clave "duracion".

```
import json

with open("archivo41.json", "r") as archivo:
    peliculas = json.load(archivo)
for pelicula in peliculas:
    if pelicula["duracion"] > 160:
        print(pelicula)
```

Ejercicio 83

Enunciado: Crear un JSON con autores y el año de publicación de su libro más famoso.
Explicación: Guardamos un diccionario donde la clave es el autor y el valor el año de publicación.

```
import json

autores = {"Shakespeare": 1603, "Cervantes": 1605, "Rowling":
1997}
with open("archivo42.json", "w") as archivo:
    json.dump(autores, archivo)
print("Archivo JSON con autores y año de publicación creado.")
```

Ejercicio 84

Enunciado: Leer un JSON y mostrar los autores con libros publicados antes de 1700.
Explicación: Iteramos sobre el diccionario y filtramos por el valor año.

```
import json

with open("archivo42.json", "r") as archivo:
    autores = json.load(archivo)
for autor, año in autores.items():
    if año < 1700:
        print(autor)
```

Ejercicio 85

Enunciado: Crear un JSON con empleados y su salario anual.
Explicación: Cada empleado es un diccionario con nombre y salario; guardamos todos en una lista.

```
import json

empleados = [
    {"nombre": "Ana", "salario": 25000},
    {"nombre": "Luis", "salario": 30000},
    {"nombre": "Marta", "salario": 28000}
]
with open("archivo43.json", "w") as archivo:
    json.dump(empleados, archivo)
print("Archivo JSON con empleados y salario creado.")
```

Ejercicio 86

Enunciado: Leer un JSON y mostrar los empleados con salario mayor a 27000.
Explicación: Iteramos sobre la lista de diccionarios y filtramos por la clave "salario".

```
import json

with open("archivo43.json", "r") as archivo:
    empleados = json.load(archivo)
for empleado in empleados:
    if empleado["salario"] > 27000:
        print(empleado)
```

Ejercicio 87

Enunciado: Crear un JSON con materias y el número de créditos de cada una.
Explicación: Guardamos un diccionario donde la clave es la materia y el valor los créditos.

```
import json

materias = {"Matemáticas": 6, "Física": 5, "Química": 4}
with open("archivo44.json", "w") as archivo:
    json.dump(materias, archivo)
print("Archivo JSON con materias y créditos creado.")
```

Ejercicio 88

Enunciado: Leer un JSON y mostrar las materias con créditos mayores a 4.
Explicación: Iteramos sobre el diccionario y filtramos por valor.

```
import json

with open("archivo44.json", "r") as archivo:
    materias = json.load(archivo)
for materia, creditos in materias.items():
    if creditos > 4:
        print(materia)
```

Ejercicio 89

Enunciado: Crear un JSON con tiendas y sus productos disponibles.
Explicación: Cada tienda tiene una lista de productos; guardamos todo en un diccionario.

```
import json

tiendas = {
    "Tienda1": ["Manzana", "Banana", "Pan"],
    "Tienda2": ["Leche", "Queso", "Pan"],
    "Tienda3": ["Cereza", "Yogurt", "Pan"]
}
with open("archivo45.json", "w") as archivo:
    json.dump(tiendas, archivo)
print("Archivo JSON con tiendas y productos creado.")
```

Ejercicio 90

Enunciado: Leer un JSON y mostrar los productos de "Tienda2".
Explicación: Accedemos al valor de la clave correspondiente en el diccionario.

```
import json

with open("archivo45.json", "r") as archivo:
    tiendas = json.load(archivo)
print("Productos de Tienda2:", tiendas["Tienda2"])
```

Ejercicio 91

Enunciado: Crear un JSON con alumnos y sus calificaciones por materia.
Explicación: Cada alumno es un diccionario con nombre y otro diccionario con materias y calificaciones.

```
import json

alumnos = [
    {"nombre": "Ana", "calificaciones": {"Matemáticas": 8,
"Física": 7}},
    {"nombre": "Luis", "calificaciones": {"Matemáticas": 6,
"Física": 9}},
    {"nombre": "Marta", "calificaciones": {"Matemáticas": 9,
"Física": 8}}
]
with open("archivo46.json", "w") as archivo:
    json.dump(alumnos, archivo)
print("Archivo JSON con alumnos y calificaciones creado.")
```

Ejercicio 92

Enunciado: Leer un JSON y mostrar las calificaciones de "Ana".
Explicación: Iteramos sobre la lista de diccionarios y accedemos al diccionario anidado de calificaciones.

```
import json

with open("archivo46.json", "r") as archivo:
    alumnos = json.load(archivo)
for alumno in alumnos:
    if alumno["nombre"] == "Ana":
        print("Calificaciones de Ana:", alumno["calificaciones"])
```

Ejercicio 93

Enunciado: Crear un JSON con películas y actores principales.
Explicación: Cada película es un diccionario con título y lista de actores.

```
import json

peliculas = [
    {"titulo": "Inception", "actores": ["Leonardo DiCaprio",
"Joseph Gordon-Levitt"]},
    {"titulo": "Titanic", "actores": ["Leonardo DiCaprio", "Kate
Winslet"]},
    {"titulo": "Avatar", "actores": ["Sam Worthington", "Zoe
Saldana"]}
]
with open("archivo47.json", "w") as archivo:
    json.dump(peliculas, archivo)
print("Archivo JSON con películas y actores creado.")
```

Ejercicio 94

Enunciado: Leer un JSON y mostrar las películas donde actúe "Leonardo DiCaprio".
Explicación: Iteramos sobre la lista y verificamos si el nombre del actor está en la lista de actores.

```
import json

with open("archivo47.json", "r") as archivo:
    peliculas = json.load(archivo)
for pelicula in peliculas:
    if "Leonardo DiCaprio" in pelicula["actores"]:
        print(pelicula["titulo"])
```

Ejercicio 95

Enunciado: Crear un JSON con ciudades y clima diario (lista de temperaturas).
Explicación: Cada ciudad tiene un diccionario con lista de temperaturas por días.

```
import json

clima = {
    "Madrid": [30, 32, 31, 29, 28],
    "Barcelona": [28, 27, 26, 29, 30]
}
with open("archivo48.json", "w") as archivo:
    json.dump(clima, archivo)
print("Archivo JSON con ciudades y clima diario creado.")
```

Ejercicio 96

Enunciado: Leer un JSON y mostrar la temperatura promedio de "Madrid".
Explicación: Calculamos la media de los valores en la lista de temperaturas.

```
import json

with open("archivo48.json", "r") as archivo:
    clima = json.load(archivo)
temperaturas_madrid = clima["Madrid"]
promedio = sum(temperaturas_madrid) / len(temperaturas_madrid)
print("Temperatura promedio de Madrid:", promedio)
```

Ejercicio 97

Enunciado: Crear un JSON con productos y su stock en diferentes almacenes.
Explicación: Cada producto es un diccionario con el nombre y otro diccionario con almacén y cantidad.

```
import json

productos = [
    {"nombre": "Manzana", "stock": {"Almacen1": 50, "Almacen2":
30}},
    {"nombre": "Leche", "stock": {"Almacen1": 20, "Almacen2": 40}}
]
with open("archivo49.json", "w") as archivo:
    json.dump(productos, archivo)
print("Archivo JSON con productos y stock creado.")
```

Ejercicio 98

Enunciado: Leer un JSON y mostrar el stock de "Leche" en "Almacen2".
Explicación: Accedemos a las claves anidadas para obtener la cantidad deseada.

```
import json

with open("archivo49.json", "r") as archivo:
    productos = json.load(archivo)
for producto in productos:
    if producto["nombre"] == "Leche":
        print("Stock de Leche en Almacen2:", producto["stock"]
["Almacen2"])
```

Ejercicio 99

Enunciado: Crear un JSON con jugadores y sus estadísticas (puntos, asistencias y rebotes).
Explicación: Cada jugador es un diccionario con nombre y otro diccionario con estadísticas.

```
import json

jugadores = [
    {"nombre": "Juan", "estadisticas": {"puntos": 20,
"asistencias": 5, "rebotes": 7}},
    {"nombre": "Luis", "estadisticas": {"puntos": 25,
"asistencias": 7, "rebotes": 10}}
]
with open("archivo50.json", "w") as archivo:
    json.dump(jugadores, archivo)
print("Archivo JSON con jugadores y estadísticas creado.")
```

Ejercicio 100

Enunciado: Leer un JSON y mostrar los jugadores con más de 22 puntos.
Explicación: Iteramos sobre la lista de diccionarios y filtramos según el valor dentro del diccionario de estadísticas.

```
import json

with open("archivo50.json", "r") as archivo:
    jugadores = json.load(archivo)
for jugador in jugadores:
    if jugador["estadisticas"]["puntos"] > 22:
        print(jugador["nombre"], "tiene", jugador["estadisticas"]
["puntos"], "puntos")
```

Ejercicios python juegos

Ejercicio 1

Enunciado: Crear un juego donde el usuario adivina un número entre 1 y 10.
Explicación: Generamos un número aleatorio y comparamos con la entrada del usuario.

```
import random

numero = random.randint(1, 10)
adivinanza = int(input("Adivina el número entre 1 y 10: "))
if adivinanza == numero:
    print("¡Correcto! Has adivinado.")
else:
    print("Incorrecto. El número era:", numero)
```

Ejercicio 2

Enunciado: Juego de piedra, papel o tijera contra la computadora.
Explicación: Comparamos la elección del usuario con la elección aleatoria de la computadora y determinamos el ganador.

```
import random

opciones = ["piedra", "papel", "tijera"]
usuario = input("Elige piedra, papel o tijera: ").lower()
computadora = random.choice(opciones)
print("Computadora eligió:", computadora)

if usuario == computadora:
    print("Empate")
elif (usuario == "piedra" and computadora == "tijera") or \
     (usuario == "papel" and computadora == "piedra") or \
     (usuario == "tijera" and computadora == "papel"):
    print("¡Ganaste!")
else:
    print("Perdiste")
```

Ejercicio 3

Enunciado: Juego de adivinar si el número generado es par o impar.
Explicación: La computadora genera un número aleatorio y el usuario predice si es par o impar.

```
import random

numero = random.randint(1, 100)
respuesta = input("¿El número es par o impar? ").lower()

if (numero % 2 == 0 and respuesta == "par") or (numero % 2 != 0
and respuesta == "impar"):
    print("¡Correcto! El número era", numero)
else:
    print("Incorrecto. El número era", numero)
```

Ejercicio 4

Enunciado: Juego de adivinar un número con intentos limitados.
Explicación: El usuario tiene 3 intentos para adivinar un número entre 1 y 5.

```
import random

numero = random.randint(1, 5)
intentos = 3

for i in range(intentos):
    adivinanza = int(input("Adivina el número entre 1 y 5: "))
    if adivinanza == numero:
        print("¡Correcto! Has adivinado.")
        break
    else:
        print("Incorrecto, intenta de nuevo.")
else:
    print("Se acabaron los intentos. El número era:", numero)
```

Ejercicio 5

Enunciado: Juego de lanzar un dado.
Explicación: La computadora simula el lanzamiento de un dado (1 a 6) y muestra el resultado.

```
import random

dado = random.randint(1, 6)
print("Has lanzado el dado y salió:", dado)
```

Ejercicio 6

Enunciado: Juego de adivinar un número con pistas de mayor o menor.
Explicación: El usuario recibe pistas para acercarse al número generado.

```
import random

numero = random.randint(1, 20)
adivinanza = 0

while adivinanza != numero:
    adivinanza = int(input("Adivina el número entre 1 y 20: "))
    if adivinanza < numero:
        print("Demasiado bajo")
    elif adivinanza > numero:
        print("Demasiado alto")
print("¡Correcto! El número era", numero)
```

Ejercicio 7

Enunciado: Juego de adivinar un color.
Explicación: La computadora tiene un color secreto y el usuario debe adivinarlo.

```
import random

colores = ["rojo", "azul", "verde", "amarillo"]
color_secreto = random.choice(colores)
adivinanza = input("Adivina el color: ").lower()

if adivinanza == color_secreto:
    print("¡Correcto!")
else:
    print("Incorrecto. El color era:", color_secreto)
```

Ejercicio 8

Enunciado: Juego de lanzar dos dados y sumar el resultado.
Explicación: Se generan dos números aleatorios entre 1 y 6 y se muestra la suma.

```
import random

dado1 = random.randint(1, 6)
dado2 = random.randint(1, 6)
suma = dado1 + dado2
print("Dado1:", dado1, "Dado2:", dado2, "Suma:", suma)
```

Ejercicio 9

Enunciado: Juego de simular lanzamiento de moneda.
Explicación: La computadora elige entre cara o cruz aleatoriamente y muestra el resultado.

```
import random

moneda = random.choice(["cara", "cruz"])
print("La moneda cayó en:", moneda)
```

Ejercicio 10

Enunciado: Juego de adivinar un número con mensaje de bienvenida y despedida.
Explicación: Se añade interacción más amigable al juego de adivinar números.

```
import random

print("¡Bienvenido al juego de adivinar números!")
numero = random.randint(1, 10)
adivinanza = int(input("Adivina un número entre 1 y 10: "))

if adivinanza == numero:
    print("¡Felicidades! Has acertado.")
else:
    print("Lo siento, el número correcto era:", numero)
print("Gracias por jugar")
```

Ejercicio 11

Enunciado: Juego de adivinar un número y dar tres pistas (mayor, menor, cercano).
Explicación: Se dan pistas más detalladas para ayudar al usuario a adivinar.

```
import random

numero = random.randint(1, 20)
adivinanza = int(input("Adivina el número entre 1 y 20: "))

if adivinanza == numero:
    print("¡Correcto!")
elif abs(adivinanza - numero) <= 2:
    print("¡Muy cerca!")
elif adivinanza < numero:
    print("Demasiado bajo")
else:
    print("Demasiado alto")
```

Ejercicio 12

Enunciado: Juego de “sí o no” aleatorio.
Explicación: La computadora responde preguntas del usuario con “Sí” o “No” de forma aleatoria.

```
import random

respuesta = random.choice(["Sí", "No"])
pregunta = input("Haz una pregunta: ")
print("Respuesta:", respuesta)
```

Ejercicio 13

Enunciado: Juego de lanzar un dado y adivinar el resultado.
Explicación: Se genera un número aleatorio y el usuario intenta adivinarlo.

```
import random

numero = random.randint(1, 6)
adivinanza = int(input("Adivina el número del dado (1-6): "))

if adivinanza == numero:
    print("¡Correcto!")
else:
    print("Incorrecto. El número era:", numero)
```

Ejercicio 14

Enunciado: Juego de adivinar si la suma de dos dados es par o impar.
Explicación: Se lanzan dos dados y el usuario predice si la suma es par o impar.

```
import random

dado1 = random.randint(1, 6)
dado2 = random.randint(1, 6)
suma = dado1 + dado2
respuesta = input("¿La suma es par o impar? ").lower()

if (suma % 2 == 0 and respuesta == "par") or (suma % 2 != 0 and
respuesta == "impar"):
    print("¡Correcto! La suma es", suma)
else:
    print("Incorrecto. La suma era", suma)
```

Ejercicio 15

Enunciado: Juego de adivinar el número secreto con intentos infinitos hasta acertar.
Explicación: Se repite la adivinanza hasta que el usuario acierte.

```
import random

numero = random.randint(1, 10)
adivinanza = 0

while adivinanza != numero:
    adivinanza = int(input("Adivina el número entre 1 y 10: "))
    if adivinanza < numero:
        print("Demasiado bajo")
    elif adivinanza > numero:
        print("Demasiado alto")

print("¡Correcto! El número era", numero)
```

Ejercicio 16

Enunciado: Juego de lanzar una moneda y adivinar el resultado.
Explicación: La computadora elige aleatoriamente cara o cruz y el usuario intenta adivinar.

```
import random

moneda = random.choice(["cara", "cruz"])
adivinanza = input("Adivina cara o cruz: ").lower()

if adivinanza == moneda:
    print("¡Correcto!")
else:
    print("Incorrecto. La moneda era", moneda)
```

Ejercicio 17

Enunciado: Juego de adivinar un número del 1 al 50 con pistas de “cerca” y “lejos”.
Explicación: Se da una pista según la cercanía de la adivinanza.

```
import random

numero = random.randint(1, 50)
adivinanza = int(input("Adivina el número entre 1 y 50: "))

if adivinanza == numero:
    print("¡Correcto!")
elif abs(adivinanza - numero) <= 5:
    print("¡Muy cerca!")
else:
    print("Lejos del número")
```

Ejercicio 18

Enunciado: Juego de adivinar un número con puntaje inicial y decremento por intentos.
Explicación: Se empieza con 10 puntos y se pierden puntos por cada intento fallido.

```
import random

numero = random.randint(1, 10)
puntaje = 10

while True:
    adivinanza = int(input("Adivina el número entre 1 y 10: "))
    if adivinanza == numero:
        print("¡Correcto! Puntaje final:", puntaje)
        break
    else:
        puntaje -= 1
        print("Incorrecto. Puntaje actual:", puntaje)
```

Ejercicio 19

Enunciado: Juego de adivinar un número y mostrar el número de intentos realizados.
Explicación: Contamos cuántos intentos necesitó el usuario para adivinar.

```
import random

numero = random.randint(1, 20)
intentos = 0
adivinanza = 0

while adivinanza != numero:
    adivinanza = int(input("Adivina el número entre 1 y 20: "))
    intentos += 1

print("¡Correcto! Número de intentos:", intentos)
```

Ejercicio 20

Enunciado: Juego de adivinar un color de una lista de colores.
Explicación: Se selecciona un color aleatorio y el usuario intenta adivinarlo.

```
import random

colores = ["rojo", "azul", "verde", "amarillo"]
color_secreto = random.choice(colores)
adivinanza = input("Adivina el color: ").lower()

if adivinanza == color_secreto:
    print("¡Correcto!")
else:
    print("Incorrecto. El color era:", color_secreto)
```

Ejercicio 21

Enunciado: Juego de adivinar un número aleatorio entre 1 y 100 con mensaje de “muy alto” o “muy bajo”.
Explicación: La computadora genera un número y el usuario recibe pistas hasta acertar.

```
import random

numero = random.randint(1, 100)
adivinanza = 0

while adivinanza != numero:
    adivinanza = int(input("Adivina un número entre 1 y 100: "))
    if adivinanza < numero:
        print("Muy bajo")
    elif adivinanza > numero:
        print("Muy alto")
print("¡Correcto! El número era", numero)
```

Ejercicio 22

Enunciado: Juego de lanzar dos dados y predecir si la suma es mayor o menor que 7.
Explicación: El usuario intenta adivinar si la suma de dos dados es mayor o menor que 7.

```
import random

dado1 = random.randint(1, 6)
dado2 = random.randint(1, 6)
suma = dado1 + dado2
respuesta = input("¿La suma es mayor o menor que 7? ").lower()

if (suma > 7 and respuesta == "mayor") or (suma < 7 and respuesta
== "menor"):
    print("¡Correcto! La suma es", suma)
else:
    print("Incorrecto. La suma era", suma)
```

Ejercicio 23

Enunciado: Juego de adivinar un número secreto con tres intentos y pistas.
Explicación: El usuario tiene solo tres intentos para adivinar un número y recibe pistas de “alto” o “bajo”.

```
import random

numero = random.randint(1, 10)
intentos = 3

for i in range(intentos):
    adivinanza = int(input("Adivina el número entre 1 y 10: "))
    if adivinanza == numero:
        print("¡Correcto!")
        break
    elif adivinanza < numero:
        print("Demasiado bajo")
    else:
        print("Demasiado alto")
else:
    print("Se acabaron los intentos. El número era:", numero)
```

Ejercicio 24

Enunciado: Juego de “Mayor o Menor” con cartas del 1 al 13.
Explicación: El usuario predice si la próxima carta será mayor o menor que la actual.

```
import random

carta_actual = random.randint(1, 13)
print("Carta actual:", carta_actual)
prediccion = input("¿La siguiente carta será mayor o menor?
").lower()
carta_siguiente = random.randint(1, 13)
print("Carta siguiente:", carta_siguiente)

if (prediccion == "mayor" and carta_siguiente > carta_actual) or \
   (prediccion == "menor" and carta_siguiente < carta_actual):
    print("¡Correcto!")
else:
    print("Incorrecto")
```

Ejercicio 25

Enunciado: Juego de adivinar un número con pistas de “muy cerca” si la diferencia ≤ 2.
Explicación: Se da una pista adicional cuando el usuario está cerca del número.

```
import random

numero = random.randint(1, 20)
adivinanza = int(input("Adivina un número entre 1 y 20: "))

if adivinanza == numero:
    print("¡Correcto!")
elif abs(adivinanza - numero) <= 2:
    print("¡Muy cerca!")
else:
    print("Incorrecto. El número era", numero)
```

Ejercicio 26

Enunciado: Juego de lanzar un dado y verificar si es par.
Explicación: La computadora genera un número y el usuario predice si es par.

```
import random

dado = random.randint(1, 6)
respuesta = input("¿El dado es par? (sí/no): ").lower()

if (dado % 2 == 0 and respuesta == "sí") or (dado % 2 != 0 and
respuesta == "no"):
    print("¡Correcto! El dado era", dado)
else:
    print("Incorrecto. El dado era", dado)
```

Ejercicio 27

Enunciado: Juego de adivinar un color secreto con pistas.
Explicación: La computadora selecciona un color y el usuario recibe pistas de “igual” o “distinto”.

```
import random

colores = ["rojo", "azul", "verde", "amarillo"]
color_secreto = random.choice(colores)
adivinanza = input("Adivina el color: ").lower()

if adivinanza == color_secreto:
    print("¡Correcto!")
else:
    print("Incorrecto. El color era", color_secreto)
```

Ejercicio 28

Enunciado: Juego de lanzar dos dados y predecir si la suma es par o impar.
Explicación: Se lanzan dos dados y el usuario predice la paridad de la suma.

```
import random

dado1 = random.randint(1, 6)
dado2 = random.randint(1, 6)
suma = dado1 + dado2
respuesta = input("¿La suma es par o impar? ").lower()

if (suma % 2 == 0 and respuesta == "par") or (suma % 2 != 0 and
respuesta == "impar"):
    print("¡Correcto! La suma es", suma)
else:
    print("Incorrecto. La suma era", suma)
```

Ejercicio 29

Enunciado: Juego de adivinar un número con límite de intentos y puntuación.
Explicación: El usuario tiene 5 intentos y empieza con 10 puntos; pierde 2 puntos por intento fallido.

```
import random

numero = random.randint(1, 10)
puntaje = 10
intentos = 5

for i in range(intentos):
    adivinanza = int(input("Adivina el número entre 1 y 10: "))
    if adivinanza == numero:
        print("¡Correcto! Puntaje:", puntaje)
        break
    else:
        puntaje -= 2
        print("Incorrecto. Puntaje actual:", puntaje)
else:
    print("Se acabaron los intentos. El número era:", numero)
```

Ejercicio 30

Enunciado: Juego de lanzar una moneda con opción de repetir.
Explicación: Permite al usuario jugar varias veces y muestra el resultado de cada lanzamiento.

```
import random

while True:
    moneda = random.choice(["cara", "cruz"])
    print("La moneda cayó en:", moneda)
    repetir = input("¿Quieres lanzar de nuevo? (sí/no): ").lower()
    if repetir != "sí":
        break
print("Gracias por jugar")
```

Ejercicio 31

Enunciado: Juego de adivinar un número secreto entre 1 y 50 con pistas “muy cerca” si la diferencia ≤ 3.
Explicación: El usuario recibe una pista adicional cuando está muy cerca del número secreto.

```
import random

numero = random.randint(1, 50)
adivinanza = int(input("Adivina el número entre 1 y 50: "))

if adivinanza == numero:
    print("¡Correcto!")
elif abs(adivinanza - numero) <= 3:
    print("¡Muy cerca!")
else:
    print("Incorrecto. El número era", numero)
```

Ejercicio 32

Enunciado: Juego de lanzar un dado y predecir si será mayor que 3.
Explicación: La computadora genera un número del 1 al 6 y el usuario predice si es mayor que 3.

```
import random

dado = random.randint(1, 6)
respuesta = input("¿El dado será mayor que 3? (sí/no): ").lower()

if (dado > 3 and respuesta == "sí") or (dado <= 3 and respuesta ==
"no"):
    print("¡Correcto! El dado era", dado)
else:
    print("Incorrecto. El dado era", dado)
```

Ejercicio 33

Enunciado: Juego de adivinar un número y mostrar si es primo.
Explicación: El usuario adivina un número y el programa indica si es primo o no.

```
numero = int(input("Ingresa un número: "))

def es_primo(n):
    if n < 2:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

if es_primo(numero):
    print("El número es primo")
else:
    print("El número no es primo")
```

Ejercicio 34

Enunciado: Juego de lanzar dos dados y predecir la mayor de las dos caras.
Explicación: La computadora genera dos números y el usuario intenta adivinar cuál es mayor.

```
import random

dado1 = random.randint(1, 6)
dado2 = random.randint(1, 6)
prediccion = int(input("Adivina cuál dado será mayor (1-6): "))

mayor = max(dado1, dado2)
print("Dado1:", dado1, "Dado2:", dado2)
if prediccion == mayor:
    print("¡Correcto!")
else:
    print("Incorrecto. El mayor era", mayor)
```

Ejercicio 35

Enunciado: Juego de adivinar un número con límite de intentos y contador de intentos restantes.
Explicación: Se muestran los intentos restantes y se finaliza cuando se acaban.

```
import random

numero = random.randint(1, 10)
intentos = 5

while intentos > 0:
    adivinanza = int(input(f"Adivina el número (Intentos
restantes: {intentos}): "))
    if adivinanza == numero:
        print("¡Correcto!")
        break
    elif adivinanza < numero:
        print("Demasiado bajo")
    else:
        print("Demasiado alto")
    intentos -= 1
else:
    print("Se acabaron los intentos. El número era:", numero)
```

Ejercicio 36

Enunciado: Juego de adivinar un color con pista de la primera letra.
Explicación: La computadora da la primera letra del color secreto como pista.

```
import random

colores = ["rojo", "azul", "verde", "amarillo"]
color_secreto = random.choice(colores)
adivinanza = input(f"Adivina el color (pista: comienza con
{color_secreto[0]}): ").lower()

if adivinanza == color_secreto:
    print("¡Correcto!")
else:
    print("Incorrecto. El color era", color_secreto)
```

Ejercicio 37

Enunciado: Juego de lanzar un dado y adivinar si será múltiplo de 3.
Explicación: La computadora genera un número aleatorio y el usuario predice si es divisible entre 3.

```
import random

dado = random.randint(1, 6)
respuesta = input("¿El dado será múltiplo de 3? (sí/no):
").lower()

if (dado % 3 == 0 and respuesta == "sí") or (dado % 3 != 0 and
respuesta == "no"):
    print("¡Correcto! El dado era", dado)
else:
    print("Incorrecto. El dado era", dado)
```

Ejercicio 38

Enunciado: Juego de adivinar un número secreto y calcular la diferencia con la respuesta.
Explicación: Se indica al usuario cuán lejos estuvo su intento.

```
import random

numero = random.randint(1, 20)
adivinanza = int(input("Adivina el número entre 1 y 20: "))
diferencia = abs(numero - adivinanza)

if adivinanza == numero:
    print("¡Correcto!")
else:
    print(f"Incorrecto. Estuviste a {diferencia} del número
secreto.")
```

Ejercicio 39

Enunciado: Juego de adivinar un número y decir si es múltiplo de 2 o 5.
Explicación: Además de adivinar, el usuario recibe información sobre múltiplos.

```
import random

numero = random.randint(1, 20)
adivinanza = int(input("Adivina el número entre 1 y 20: "))

if adivinanza == numero:
    print("¡Correcto!")
else:
    print("Incorrecto. El número era", numero)
    if numero % 2 == 0:
        print("Es múltiplo de 2")
    if numero % 5 == 0:
        print("Es múltiplo de 5")
```

Ejercicio 40

Enunciado: Juego de adivinar un número y mostrar mensaje final según la cercanía.
Explicación: Se da retroalimentación final sobre la aproximación del usuario.

```
import random

numero = random.randint(1, 20)
adivinanza = int(input("Adivina el número entre 1 y 20: "))
diferencia = abs(numero - adivinanza)

if adivinanza == numero:
    print("¡Exacto! Has adivinado.")
elif diferencia <= 3:
    print("¡Muy cerca! El número era", numero)
else:
    print("Lejos del número. El número era", numero)
```

Ejercicio 41

Enunciado: Juego de adivinar un número secreto entre 1 y 30 con pistas “muy cerca” si la diferencia ≤ 2.
Explicación: Se da una pista adicional cuando la adivinanza está muy cerca del número secreto.

```
import random

numero = random.randint(1, 30)
adivinanza = int(input("Adivina el número entre 1 y 30: "))

if adivinanza == numero:
    print("¡Correcto!")
elif abs(adivinanza - numero) <= 2:
    print("¡Muy cerca!")
else:
    print("Incorrecto. El número era", numero)
```

Ejercicio 42

Enunciado: Juego de adivinar la suma de dos dados.
Explicación: El usuario intenta adivinar la suma de dos números generados aleatoriamente.

```
import random

dado1 = random.randint(1, 6)
dado2 = random.randint(1, 6)
suma = dado1 + dado2
adivinanza = int(input("Adivina la suma de dos dados (2-12): "))

if adivinanza == suma:
    print("¡Correcto!")
else:
    print("Incorrecto. La suma era", suma)
```

Ejercicio 43

Enunciado: Juego de adivinar un número y dar retroalimentación “alto” o “bajo” hasta acertar.
Explicación: Se usa un bucle hasta que el usuario acierte el número secreto.

```
import random

numero = random.randint(1, 20)
adivinanza = 0

while adivinanza != numero:
    adivinanza = int(input("Adivina un número entre 1 y 20: "))
    if adivinanza < numero:
        print("Demasiado bajo")
    elif adivinanza > numero:
        print("Demasiado alto")
print("¡Correcto! El número era", numero)
```

Ejercicio 44

Enunciado: Juego de lanzar una moneda y adivinar “cara” o “cruz”.
Explicación: La computadora selecciona aleatoriamente y se compara con la elección del usuario.

```
import random

moneda = random.choice(["cara", "cruz"])
adivinanza = input("Adivina cara o cruz: ").lower()

if adivinanza == moneda:
    print("¡Correcto!")
else:
    print("Incorrecto. La moneda era", moneda)
```

Ejercicio 45

Enunciado: Juego de adivinar un color secreto de una lista de colores.
Explicación: La computadora selecciona un color y el usuario intenta adivinarlo.

```
import random

colores = ["rojo", "azul", "verde", "amarillo", "naranja"]
color_secreto = random.choice(colores)
adivinanza = input("Adivina el color: ").lower()

if adivinanza == color_secreto:
    print("¡Correcto!")
else:
    print("Incorrecto. El color era", color_secreto)
```

Ejercicio 46

Enunciado: Juego de lanzar un dado y predecir si el resultado es par o impar.
Explicación: El usuario predice la paridad del número generado aleatoriamente.

```
import random

dado = random.randint(1, 6)
respuesta = input("¿El dado es par o impar? ").lower()

if (dado % 2 == 0 and respuesta == "par") or (dado % 2 != 0 and
respuesta == "impar"):
    print("¡Correcto! El dado era", dado)
else:
    print("Incorrecto. El dado era", dado)
```

Ejercicio 47

Enunciado: Juego de adivinar un número secreto con puntuación inicial y decremento por intentos fallidos.
Explicación: El usuario empieza con 10 puntos y pierde 2 puntos por cada intento incorrecto.

```
import random

numero = random.randint(1, 10)
puntaje = 10

while True:
    adivinanza = int(input("Adivina un número entre 1 y 10: "))
    if adivinanza == numero:
        print("¡Correcto! Puntaje final:", puntaje)
        break
    else:
        puntaje -= 2
        print("Incorrecto. Puntaje actual:", puntaje)
```

Ejercicio 48

Enunciado: Juego de adivinar un número con límite de intentos y mensaje final según desempeño.
Explicación: Se dan mensajes distintos según los intentos usados para adivinar correctamente.

```
import random

numero = random.randint(1, 20)
intentos = 5

for i in range(intentos):
    adivinanza = int(input(f"Adivina el número (Intentos
restantes: {intentos - i}): "))
    if adivinanza == numero:
        print("¡Correcto!")
        break
else:
    print("Se acabaron los intentos. El número era:", numero)
```

Ejercicio 49

Enunciado: Juego de adivinar un número y mostrar si es múltiplo de 2, 3 o 5.
Explicación: Además de adivinar, se indican los múltiplos del número secreto.

```
import random

numero = random.randint(1, 20)
adivinanza = int(input("Adivina un número entre 1 y 20: "))

if adivinanza == numero:
    print("¡Correcto!")
else:
    print("Incorrecto. El número era", numero)
    if numero % 2 == 0:
        print("Es múltiplo de 2")
    if numero % 3 == 0:
        print("Es múltiplo de 3")
    if numero % 5 == 0:
        print("Es múltiplo de 5")
```

Ejercicio 50

Enunciado: Juego de adivinar un número y mostrar la diferencia absoluta con la respuesta del usuario.
Explicación: Se indica al usuario cuán lejos estuvo del número secreto.

```
import random

numero = random.randint(1, 20)
adivinanza = int(input("Adivina un número entre 1 y 20: "))
diferencia = abs(numero - adivinanza)

if adivinanza == numero:
    print("¡Exacto! Has adivinado.")
else:
    print(f"Incorrecto. Estuviste a {diferencia} del número
secreto.")
```

Ejercicio 51

Enunciado: Juego de adivinar un número secreto con pistas de “caliente” si la diferencia ≤ 5.
Explicación: Se da una pista adicional al usuario cuando su adivinanza está cerca del número secreto.

```
import random

numero = random.randint(1, 50)
adivinanza = int(input("Adivina un número entre 1 y 50: "))

if adivinanza == numero:
    print("¡Correcto!")
elif abs(adivinanza - numero) <= 5:
    print("¡Caliente!")
else:
    print("¡Fríо! El número era", numero)
```

Ejercicio 52

Enunciado: Juego de lanzar un dado y predecir si es mayor o menor que 4.
Explicación: El usuario intenta adivinar si el número generado es mayor o menor que 4.

```
import random

dado = random.randint(1, 6)
respuesta = input("¿El dado será mayor que 4? (sí/no): ").lower()

if (dado > 4 and respuesta == "sí") or (dado <= 4 and respuesta ==
"no"):
    print("¡Correcto! El dado era", dado)
else:
    print("Incorrecto. El dado era", dado)
```

Ejercicio 53

Enunciado: Juego de adivinar un color secreto de una lista con opción de reintento.
Explicación: El usuario puede intentar varias veces hasta adivinar el color.

```
import random

colores = ["rojo", "azul", "verde", "amarillo", "naranja"]
color_secreto = random.choice(colores)
adivinanza = ""

while adivinanza != color_secreto:
    adivinanza = input("Adivina el color: ").lower()
    if adivinanza == color_secreto:
        print("¡Correcto!")
    else:
        print("Incorrecto. Intenta de nuevo.")
```

Ejercicio 54

Enunciado: Juego de adivinar un número con mensaje de “cerca” si la diferencia ≤ 3.
Explicación: Se da retroalimentación para acercar al usuario al número secreto.

```
import random

numero = random.randint(1, 30)
adivinanza = int(input("Adivina el número entre 1 y 30: "))

if adivinanza == numero:
    print("¡Correcto!")
elif abs(adivinanza - numero) <= 3:
    print("¡Cerca!")
else:
    print("Lejos. El número era", numero)
```

Ejercicio 55

Enunciado: Juego de lanzar dos dados y predecir si la suma es mayor o menor que 8.
Explicación: El usuario intenta adivinar la suma de dos dados en comparación con 8.

```
import random

dado1 = random.randint(1, 6)
dado2 = random.randint(1, 6)
suma = dado1 + dado2
respuesta = input("¿La suma es mayor o menor que 8? ").lower()

if (suma > 8 and respuesta == "mayor") or (suma <= 8 and respuesta
== "menor"):
    print("¡Correcto! La suma era", suma)
else:
    print("Incorrecto. La suma era", suma)
```

Ejercicio 56

Enunciado: Juego de adivinar un número con conteo de intentos.
Explicación: Se indica cuántos intentos tomó al usuario adivinar el número.

```
import random

numero = random.randint(1, 20)
intentos = 0
adivinanza = 0

while adivinanza != numero:
    adivinanza = int(input("Adivina un número entre 1 y 20: "))
    intentos += 1

print("¡Correcto! Número de intentos:", intentos)
```

Ejercicio 57

Enunciado: Juego de lanzar una moneda y repetir hasta acertar.
Explicación: El usuario puede seguir jugando hasta predecir correctamente “cara” o “cruz”.

```
import random

while True:
    moneda = random.choice(["cara", "cruz"])
    adivinanza = input("Adivina cara o cruz: ").lower()
    if adivinanza == moneda:
        print("¡Correcto!")
        break
    else:
        print("Incorrecto. Intenta de nuevo.")
```

Ejercicio 58

Enunciado: Juego de adivinar un número y mostrar si es divisible entre 2 o 5.
Explicación: Se indica al usuario información extra sobre múltiplos del número secreto.

```
import random

numero = random.randint(1, 20)
adivinanza = int(input("Adivina el número entre 1 y 20: "))

if adivinanza == numero:
    print("¡Correcto!")
else:
    print("Incorrecto. El número era", numero)
    if numero % 2 == 0:
        print("Es divisible entre 2")
    if numero % 5 == 0:
        print("Es divisible entre 5")
```

Ejercicio 59

Enunciado: Juego de adivinar un número con mensaje final según cercanía.
Explicación: Se da feedback al usuario según la diferencia con el número secreto.

```
import random

numero = random.randint(1, 20)
adivinanza = int(input("Adivina un número entre 1 y 20: "))
diferencia = abs(numero - adivinanza)

if adivinanza == numero:
    print("¡Exacto! Has adivinado.")
elif diferencia <= 3:
    print("¡Muy cerca! El número era", numero)
else:
    print("Lejos. El número era", numero)
```

Ejercicio 60

Enunciado: Juego de adivinar un color con pista de longitud de la palabra.
Explicación: Se da una pista indicando cuántas letras tiene el color secreto.

```
import random

colores = ["rojo", "azul", "verde", "amarillo", "naranja"]
color_secreto = random.choice(colores)
adivinanza = input(f"Adivina el color (pista: tiene
{len(color_secreto)} letras): ").lower()

if adivinanza == color_secreto:
    print("¡Correcto!")
else:
    print("Incorrecto. El color era", color_secreto)
```

Ejercicio 61

Enunciado: Juego de adivinar un número secreto con retroalimentación “muy cerca” si la diferencia ≤ 4.
Explicación: Se da una pista adicional cuando la adivinanza está cerca del número secreto.

```
import random

numero = random.randint(1, 50)
adivinanza = int(input("Adivina un número entre 1 y 50: "))

if adivinanza == numero:
    print("¡Correcto!")
elif abs(adivinanza - numero) <= 4:
    print("¡Muy cerca!")
else:
    print("Lejos. El número era", numero)
```

Ejercicio 62

Enunciado: Juego de lanzar un dado y adivinar si es múltiplo de 2 o 3.
Explicación: El usuario intenta predecir si el número generado es divisible entre 2 o 3.

```
import random

dado = random.randint(1, 6)
respuesta = input("¿El dado será múltiplo de 2 o 3? (sí/no):
").lower()

if (dado % 2 == 0 or dado % 3 == 0) and respuesta == "sí":
    print("¡Correcto! El dado era", dado)
elif (dado % 2 != 0 and dado % 3 != 0) and respuesta == "no":
    print("¡Correcto! El dado era", dado)
else:
    print("Incorrecto. El dado era", dado)
```

Ejercicio 63

Enunciado: Juego de adivinar un color secreto con pistas de vocales presentes.
Explicación: Se da una pista sobre las vocales contenidas en el color secreto.

```
import random

colores = ["rojo", "azul", "verde", "amarillo", "naranja"]
color_secreto = random.choice(colores)
vocales = [letra for letra in color_secreto if letra in "aeiou"]
adivinanza = input(f"Adivina el color (pista: contiene las vocales
{''.join(vocales)}): ").lower()

if adivinanza == color_secreto:
    print("¡Correcto!")
else:
    print("Incorrecto. El color era", color_secreto)
```

Ejercicio 64

Enunciado: Juego de adivinar un número secreto con límite de intentos y conteo regresivo.
Explicación: Se muestran los intentos restantes y se finaliza al agotarlos.

```
import random

numero = random.randint(1, 20)
intentos = 5

while intentos > 0:
    adivinanza = int(input(f"Adivina el número (Intentos
restantes: {intentos}): "))
    if adivinanza == numero:
        print("¡Correcto!")
        break
    elif adivinanza < numero:
        print("Demasiado bajo")
    else:
        print("Demasiado alto")
    intentos -= 1
else:
    print("Se acabaron los intentos. El número era", numero)
```

Ejercicio 65

Enunciado: Juego de lanzar dos dados y predecir si la suma es par o impar.
Explicación: El usuario predice la paridad de la suma de dos números generados aleatoriamente.

```
import random

dado1 = random.randint(1, 6)
dado2 = random.randint(1, 6)
suma = dado1 + dado2
respuesta = input("¿La suma es par o impar? ").lower()

if (suma % 2 == 0 and respuesta == "par") or (suma % 2 != 0 and
respuesta == "impar"):
    print("¡Correcto! La suma era", suma)
else:
    print("Incorrecto. La suma era", suma)
```

Ejercicio 66

Enunciado: Juego de adivinar un número y mostrar si es divisible entre 2, 3 o 5.
Explicación: Además de adivinar, se indica si el número secreto es divisible entre 2, 3 o 5.

```
import random

numero = random.randint(1, 30)
adivinanza = int(input("Adivina un número entre 1 y 30: "))

if adivinanza == numero:
    print("¡Correcto!")
else:
    print("Incorrecto. El número era", numero)
    if numero % 2 == 0:
        print("Divisible entre 2")
    if numero % 3 == 0:
        print("Divisible entre 3")
    if numero % 5 == 0:
        print("Divisible entre 5")
```

Ejercicio 67

Enunciado: Juego de adivinar un color secreto con pista de la primera y última letra.
Explicación: Se da una pista indicando la primera y última letra del color secreto.

```
import random

colores = ["rojo", "azul", "verde", "amarillo", "naranja"]
color_secreto = random.choice(colores)
adivinanza = input(f"Adivina el color (pista: empieza con
{color_secreto[0]} y termina con {color_secreto[-1]}): ").lower()

if adivinanza == color_secreto:
    print("¡Correcto!")
else:
    print("Incorrecto. El color era", color_secreto)
```

Ejercicio 68

Enunciado: Juego de adivinar un número secreto con puntaje inicial y decremento por intentos.
Explicación: Se empieza con 10 puntos y se pierden 1 por cada intento fallido hasta acertar.

```
import random

numero = random.randint(1, 20)
puntaje = 10

while True:
    adivinanza = int(input("Adivina un número entre 1 y 20: "))
    if adivinanza == numero:
        print("¡Correcto! Puntaje final:", puntaje)
        break
    else:
        puntaje -= 1
        print("Incorrecto. Puntaje actual:", puntaje)
```

Ejercicio 69

Enunciado: Juego de lanzar un dado hasta adivinar un número predicho.
Explicación: El usuario predice un número y se repite el lanzamiento hasta que coincida.

```
import random

prediccion = int(input("Predice un número del dado (1-6): "))
while True:
    dado = random.randint(1, 6)
    print("El dado cayó en:", dado)
    if dado == prediccion:
        print("¡Has acertado!")
        break
```

Ejercicio 70

Enunciado: Juego de adivinar un número secreto con pista “caliente” o “frío” según la diferencia.
Explicación: Se da feedback basado en la proximidad de la adivinanza al número secreto.

```
import random

numero = random.randint(1, 50)
adivinanza = int(input("Adivina un número entre 1 y 50: "))
diferencia = abs(numero - adivinanza)

if adivinanza == numero:
    print("¡Exacto!")
elif diferencia <= 5:
    print("¡Caliente!")
else:
    print("¡Frío! El número era", numero)
```

Ejercicio 71

Enunciado: Juego de adivinar un número secreto con pistas de “muy cerca” si la diferencia ≤ 3.
Explicación: Se da una pista adicional al usuario cuando la adivinanza está cerca del número secreto.

```
import random

numero = random.randint(1, 50)
adivinanza = int(input("Adivina un número entre 1 y 50: "))

if adivinanza == numero:
    print("¡Correcto!")
elif abs(adivinanza - numero) <= 3:
    print("¡Muy cerca!")
else:
    print("Lejos. El número era", numero)
```

Ejercicio 72

Enunciado: Juego de lanzar un dado y predecir si es mayor que 3.
Explicación: El usuario intenta adivinar si el número generado es mayor que 3.

```
import random

dado = random.randint(1, 6)
respuesta = input("¿El dado será mayor que 3? (sí/no): ").lower()

if (dado > 3 and respuesta == "sí") or (dado <= 3 and respuesta ==
"no"):
    print("¡Correcto! El dado era", dado)
else:
    print("Incorrecto. El dado era", dado)
```

Ejercicio 73

Enunciado: Juego de adivinar un color secreto con pistas de vocales presentes.
Explicación: Se da una pista sobre las vocales contenidas en el color secreto.

```
import random

colores = ["rojo", "azul", "verde", "amarillo", "naranja"]
color_secreto = random.choice(colores)
vocales = [letra for letra in color_secreto if letra in "aeiou"]
adivinanza = input(f"Adivina el color (pista: contiene las vocales
{''.join(vocales)}): ").lower()

if adivinanza == color_secreto:
    print("¡Correcto!")
else:
    print("Incorrecto. El color era", color_secreto)
```

Ejercicio 74

Enunciado: Juego de adivinar un número con límite de intentos y conteo regresivo.
Explicación: Se muestran los intentos restantes y se finaliza al agotarlos.

```
import random

numero = random.randint(1, 20)
intentos = 5

while intentos > 0:
    adivinanza = int(input(f"Adivina el número (Intentos
restantes: {intentos}): "))
    if adivinanza == numero:
        print("¡Correcto!")
        break
    elif adivinanza < numero:
        print("Demasiado bajo")
    else:
        print("Demasiado alto")
    intentos -= 1
else:
    print("Se acabaron los intentos. El número era", numero)
```

Ejercicio 75

Enunciado: Juego de lanzar dos dados y predecir si la suma es par o impar.
Explicación: El usuario predice la paridad de la suma de dos números generados aleatoriamente.

```
import random

dado1 = random.randint(1, 6)
dado2 = random.randint(1, 6)
suma = dado1 + dado2
respuesta = input("¿La suma es par o impar? ").lower()

if (suma % 2 == 0 and respuesta == "par") or (suma % 2 != 0 and
respuesta == "impar"):
    print("¡Correcto! La suma era", suma)
else:
    print("Incorrecto. La suma era", suma)
```

Ejercicio 76

Enunciado: Juego de adivinar un número secreto con mensaje de “caliente” o “frío” según la diferencia.

Explicación: Se da feedback basado en la proximidad de la adivinanza al número secreto.

```
import random

numero = random.randint(1, 50)
adivinanza = int(input("Adivina un número entre 1 y 50: "))
diferencia = abs(numero - adivinanza)

if adivinanza == numero:
    print("¡Exacto!")
elif diferencia <= 5:
    print("¡Caliente!")
else:
    print("¡Fríо! El número era", numero)
```

Ejercicio 77

Enunciado: Juego de adivinar un color secreto con pista de longitud de la palabra.

Explicación: Se da una pista indicando cuántas letras tiene el color secreto.

```
import random

colores = ["rojo", "azul", "verde", "amarillo", "naranja"]
color_secreto = random.choice(colores)
adivinanza = input(f"Adivina el color (pista: tiene
{len(color_secreto)} letras): ").lower()

if adivinanza == color_secreto:
    print("¡Correcto!")
else:
    print("Incorrecto. El color era", color_secreto)
```

Ejercicio 78

Enunciado: Juego de adivinar un número con límite de intentos y puntaje inicial de 10.
Explicación: Se pierden 2 puntos por cada intento incorrecto hasta acertar.

```
import random

numero = random.randint(1, 20)
puntaje = 10
intentos = 5

for i in range(intentos):
    adivinanza = int(input(f"Adivina el número (Intentos
restantes: {intentos - i}): "))
    if adivinanza == numero:
        print("¡Correcto! Puntaje final:", puntaje)
        break
    else:
        puntaje -= 2
        print("Incorrecto. Puntaje actual:", puntaje)
else:
    print("Se acabaron los intentos. El número era", numero)
```

Ejercicio 79

Enunciado: Juego de lanzar un dado hasta adivinar un número predicho.
Explicación: El usuario predice un número y se repite el lanzamiento hasta que coincida.

```
import random

prediccion = int(input("Predice un número del dado (1-6): "))
while True:
    dado = random.randint(1, 6)
    print("El dado cayó en:", dado)
    if dado == prediccion:
        print("¡Has acertado!")
        break
```

Ejercicio 80

Enunciado: Juego de adivinar un número secreto con pistas “frío”, “cálido” y “caliente”.
Explicación: Se da feedback según qué tan lejos o cerca está la adivinanza del número secreto.

```
import random

numero = random.randint(1, 50)
adivinanza = int(input("Adivina un número entre 1 y 50: "))
diferencia = abs(numero - adivinanza)

if adivinanza == numero:
    print("¡Exacto!")
elif diferencia <= 3:
    print("¡Caliente!")
elif diferencia <= 7:
    print("¡Cálido!")
else:
    print("¡Frío! El número era", numero)
```

Ejercicio 81

Enunciado: Juego de adivinar un número secreto con pista de “muy frío” si la diferencia ≥ 15.
Explicación: Se da una retroalimentación indicando si la adivinanza está muy lejos del número secreto.

```
import random

numero = random.randint(1, 50)
adivinanza = int(input("Adivina un número entre 1 y 50: "))
diferencia = abs(numero - adivinanza)

if adivinanza == numero:
    print("¡Correcto!")
elif diferencia >= 15:
    print("¡Muy frío!")
else:
    print("No es correcto. El número era", numero)
```

Ejercicio 82

Enunciado: Juego de lanzar un dado y adivinar si es mayor, menor o igual a 3.
Explicación: El usuario predice si el dado es mayor, menor o igual a 3.

```
import random

dado = random.randint(1, 6)
respuesta = input("¿El dado es mayor, menor o igual a 3?
").lower()

if (dado > 3 and respuesta == "mayor") or (dado < 3 and respuesta
== "menor") or (dado == 3 and respuesta == "igual"):
    print("¡Correcto! El dado era", dado)
else:
    print("Incorrecto. El dado era", dado)
```

Ejercicio 83

Enunciado: Juego de adivinar un color secreto de una lista con opción de pistas de longitud y primera letra.
Explicación: Se ofrecen pistas combinadas para ayudar al usuario a adivinar.

```
import random

colores = ["rojo", "azul", "verde", "amarillo", "naranja"]
color_secreto = random.choice(colores)
adivinanza = input(f"Adivina el color (pista: empieza con
{color_secreto[0]} y tiene {len(color_secreto)} letras):
").lower()

if adivinanza == color_secreto:
    print("¡Correcto!")
else:
    print("Incorrecto. El color era", color_secreto)
```

Ejercicio 84

Enunciado: Juego de adivinar un número con límite de intentos y mensaje de “demasiado bajo” o “demasiado alto”.
Explicación: Se da retroalimentación inmediata para guiar al usuario hacia la respuesta correcta.

```
import random

numero = random.randint(1, 20)
intentos = 5

for i in range(intentos):
    adivinanza = int(input(f"Adivina un número (Intentos
restantes: {intentos - i}): "))
    if adivinanza == numero:
        print("¡Correcto!")
        break
    elif adivinanza < numero:
        print("Demasiado bajo")
    else:
        print("Demasiado alto")
else:
    print("Se acabaron los intentos. El número era", numero)
```

Ejercicio 85

Enunciado: Juego de lanzar dos dados y predecir si la suma es múltiplo de 2, 3 o 5.
Explicación: Se ofrece información sobre divisibilidad para que el usuario haga su predicción.

```
import random

dado1 = random.randint(1, 6)
dado2 = random.randint(1, 6)
suma = dado1 + dado2
respuesta = input("¿La suma es múltiplo de 2, 3 o 5? (sí/no):
").lower()

if (suma % 2 == 0 or suma % 3 == 0 or suma % 5 == 0) and respuesta
== "sí":
    print("¡Correcto! La suma era", suma)
elif (suma % 2 != 0 and suma % 3 != 0 and suma % 5 != 0) and
respuesta == "no":
    print("¡Correcto! La suma era", suma)
else:
    print("Incorrecto. La suma era", suma)
```

Ejercicio 86

Enunciado: Juego de adivinar un número y mostrar un mensaje según la cercanía: frío, cálido o caliente.
Explicación: Se dan pistas según qué tan cerca está la adivinanza del número secreto.

```
import random

numero = random.randint(1, 50)
adivinanza = int(input("Adivina un número entre 1 y 50: "))
diferencia = abs(numero - adivinanza)

if adivinanza == numero:
    print("¡Exacto!")
elif diferencia <= 3:
    print("¡Caliente!")
elif diferencia <= 7:
    print("¡Cálido!")
else:
    print("¡Fríо! El número era", numero)
```

Ejercicio 87

Enunciado: Juego de lanzar un dado y adivinar si el resultado será par.
Explicación: El usuario predice la paridad del número generado aleatoriamente.

```
import random

dado = random.randint(1, 6)
respuesta = input("¿El dado es par? (sí/no): ").lower()

if (dado % 2 == 0 and respuesta == "sí") or (dado % 2 != 0 and
respuesta == "no"):
    print("¡Correcto! El dado era", dado)
else:
    print("Incorrecto. El dado era", dado)
```

Ejercicio 88

Enunciado: Juego de adivinar un color secreto con pista de consonantes presentes.
Explicación: Se da una pista mostrando las consonantes contenidas en el color secreto.

```
import random

colores = ["rojo", "azul", "verde", "amarillo", "naranja"]
color_secreto = random.choice(colores)
consonantes = [letra for letra in color_secreto if letra not in
"aeiou"]
adivinanza = input(f"Adivina el color (pista: contiene las
consonantes {''.join(consonantes)}): ").lower()

if adivinanza == color_secreto:
    print("¡Correcto!")
else:
    print("Incorrecto. El color era", color_secreto)
```

Ejercicio 89

Enunciado: Juego de adivinar un número con puntaje inicial y decremento por intentos fallidos.
Explicación: Se empieza con 10 puntos y se pierden 1 por cada intento fallido hasta acertar.

```
import random

numero = random.randint(1, 20)
puntaje = 10

while True:
    adivinanza = int(input("Adivina un número entre 1 y 20: "))
    if adivinanza == numero:
        print("¡Correcto! Puntaje final:", puntaje)
        break
    else:
        puntaje -= 1
        print("Incorrecto. Puntaje actual:", puntaje)
```

Ejercicio 90

Enunciado: Juego de lanzar un dado hasta adivinar un número predicho por el usuario.
Explicación: Se repite el lanzamiento hasta que el número del dado coincida con la predicción del usuario.

```
import random

prediccion = int(input("Predice un número del dado (1-6): "))
while True:
    dado = random.randint(1, 6)
    print("El dado cayó en:", dado)
    if dado == prediccion:
        print("¡Has acertado!")
        break
```

Ejercicio 91

Enunciado: Juego de adivinar un número secreto con pistas “muy caliente” si la diferencia ≤ 2.
Explicación: Se da una pista extra para adivinanzas muy cercanas al número secreto.

```
import random

numero = random.randint(1, 50)
adivinanza = int(input("Adivina un número entre 1 y 50: "))
diferencia = abs(numero - adivinanza)

if adivinanza == numero:
    print("¡Exacto!")
elif diferencia <= 2:
    print("¡Muy caliente!")
elif diferencia <= 5:
    print("¡Caliente!")
else:
    print("¡Frío! El número era", numero)
```

Ejercicio 92

Enunciado: Juego de lanzar un dado y adivinar si es múltiplo de 2, 3 o 5.
Explicación: Se predice si el dado es divisible entre 2, 3 o 5.

```
import random

dado = random.randint(1, 6)
respuesta = input("¿El dado es múltiplo de 2, 3 o 5? (sí/no):
").lower()

if (dado % 2 == 0 or dado % 3 == 0 or dado % 5 == 0) and respuesta
== "sí":
    print("¡Correcto! El dado era", dado)
elif (dado % 2 != 0 and dado % 3 != 0 and dado % 5 != 0) and
respuesta == "no":
    print("¡Correcto! El dado era", dado)
else:
    print("Incorrecto. El dado era", dado)
```

Ejercicio 93

Enunciado: Juego de adivinar un color secreto con pista de primera y última letra y longitud.
Explicación: Se combinan varias pistas para ayudar al usuario a adivinar el color.

```
import random

colores = ["rojo", "azul", "verde", "amarillo", "naranja"]
color_secreto = random.choice(colores)
adivinanza = input(f"Adivina el color (pista: empieza con
{color_secreto[0]}, termina con {color_secreto[-1]} y tiene
{len(color_secreto)} letras): ").lower()

if adivinanza == color_secreto:
    print("¡Correcto!")
else:
    print("Incorrecto. El color era", color_secreto)
```

Ejercicio 94

Enunciado: Juego de adivinar un número con límite de intentos y mensaje “demasiado alto” o “demasiado bajo”.
Explicación: Se da retroalimentación inmediata para guiar al usuario.

```
import random

numero = random.randint(1, 20)
intentos = 5

for i in range(intentos):
    adivinanza = int(input(f"Adivina un número (Intentos
restantes: {intentos - i}): "))
    if adivinanza == numero:
        print("¡Correcto!")
        break
    elif adivinanza < numero:
        print("Demasiado bajo")
    else:
        print("Demasiado alto")
else:
    print("Se acabaron los intentos. El número era", numero)
```

Ejercicio 95

Enunciado: Juego de lanzar dos dados y adivinar si la suma es mayor que 7.
Explicación: Se compara la suma de dos dados con 7.

```
import random

dado1 = random.randint(1, 6)
dado2 = random.randint(1, 6)
suma = dado1 + dado2
respuesta = input("¿La suma es mayor que 7? (sí/no): ").lower()

if (suma > 7 and respuesta == "sí") or (suma <= 7 and respuesta ==
"no"):
    print("¡Correcto! La suma era", suma)
else:
    print("Incorrecto. La suma era", suma)
```

Ejercicio 96

Enunciado: Juego de adivinar un número con mensaje “frío”, “cálido” o “caliente”.
Explicación: Se da feedback según la cercanía de la adivinanza al número secreto.

```
import random

numero = random.randint(1, 50)
adivinanza = int(input("Adivina un número entre 1 y 50: "))
diferencia = abs(numero - adivinanza)

if adivinanza == numero:
    print("¡Correcto!")
elif diferencia <= 3:
    print("¡Caliente!")
elif diferencia <= 7:
    print("¡Cálido!")
else:
    print("¡Fríо! El número era", numero)
```

Ejercicio 97

Enunciado: Juego de lanzar un dado y predecir si el número será menor que 4.
Explicación: El usuario predice si el dado cae en un número menor que 4.

```
import random

dado = random.randint(1, 6)
respuesta = input("¿El dado es menor que 4? (sí/no): ").lower()

if (dado < 4 and respuesta == "sí") or (dado >= 4 and respuesta ==
"no"):
    print("¡Correcto! El dado era", dado)
else:
    print("Incorrecto. El dado era", dado)
```

Ejercicio 98

Enunciado: Juego de adivinar un color secreto con pista de vocales y consonantes.
Explicación: Se dan pistas indicando vocales y consonantes presentes en el color secreto.

```
import random

colores = ["rojo", "azul", "verde", "amarillo", "naranja"]
color_secreto = random.choice(colores)
vocales = [letra for letra in color_secreto if letra in "aeiou"]
consonantes = [letra for letra in color_secreto if letra not in
"aeiou"]
adivinanza = input(f"Adivina el color (vocales:
{''.join(vocales)}, consonantes: {''.join(consonantes)}):
").lower()

if adivinanza == color_secreto:
    print("¡Correcto!")
else:
    print("Incorrecto. El color era", color_secreto)
```

Ejercicio 99

Enunciado: Juego de adivinar un número con puntaje inicial y decremento por intentos.
Explicación: Se empieza con 10 puntos y se pierde 1 punto por cada intento fallido hasta acertar.

```
import random

numero = random.randint(1, 20)
puntaje = 10

while True:
    adivinanza = int(input("Adivina un número entre 1 y 20: "))
    if adivinanza == numero:
        print("¡Correcto! Puntaje final:", puntaje)
        break
    else:
        puntaje -= 1
        print("Incorrecto. Puntaje actual:", puntaje)
```

Ejercicio 100

Enunciado: Juego de lanzar un dado hasta adivinar el número predicho por el usuario.
Explicación: Se repite el lanzamiento hasta que el número del dado coincida con la predicción.

```
import random

prediccion = int(input("Predice un número del dado (1-6): "))
while True:
    dado = random.randint(1, 6)
    print("El dado cayó en:", dado)
    if dado == prediccion:
        print("¡Has acertado!")
        break
```

Epílogo

Llegar al final de este libro no es solo un cierre; es un punto de partida. Cada ejercicio resuelto, cada línea de código escrita y cada concepto comprendido representan un paso más en tu camino como programador. La programación, y en particular Python, es un lenguaje que se aprende haciendo, explorando y experimentando, y espero que este libro haya servido como un apoyo sólido en ese proceso.

Más allá de dominar la sintaxis o resolver problemas específicos, aprender a programar fomenta habilidades que trascienden el mundo del código: la capacidad de pensar de manera lógica, estructurada y analítica, la creatividad para encontrar soluciones innovadoras y la perseverancia para superar desafíos. Python, con su simplicidad y versatilidad, te permitirá aplicar estas habilidades en infinidad de proyectos, desde el desarrollo de aplicaciones y análisis de datos hasta la creación de algoritmos y la inteligencia artificial.

Quiero aprovechar este momento para reiterar mi agradecimiento a mi familia, cuyo apoyo constante ha sido un pilar fundamental en la creación de esta obra. Su confianza y aliento me han permitido dedicar tiempo y energía a compartir conocimientos, con la esperanza de que otros puedan beneficiarse de ellos y crecer en el mundo de la programación.

Mi deseo es que cada lector que haya trabajado con este libro sienta la motivación de seguir explorando, aprendiendo y creando. La programación es un viaje continuo, y cada proyecto, cada error y cada éxito son oportunidades para aprender y mejorar. Que este libro sea solo el inicio de un camino lleno de descubrimientos, retos y satisfacciones en el apasionante mundo de Python.

Recuerda siempre: la práctica constante y la curiosidad son tus mejores aliados. No hay límites para lo que puedes lograr si sigues aprendiendo, explorando y, sobre todo, creando.